CSE 475: Statistical Methods in AI **Monsoon 2019**

SMAI-M-2019 1: Mathematical Foundations of Machine Leaning - I

Lecturer: C. V. Jawahar *Date: 29 July 2019*

1.1 About the Course: SMAI

Machine learning has taken the central stage of modern artificial intelligence (AI). Data driven intelligence has found deep inroads into various walks of our life. Such techniques have found great success in diverse problems in perception (such as natural language processing, computer vision, speech understanding), control (robotics, automation) to computer systems (networks, operating systems, architecture design) and interdisciplinary domains (like biology, environmental science etc.). This makes this sub-discipline of modern AI, i.e., statistical methods in AI (SMAI), critical to the research in many areas of wider interest to IIIT-H, and the society around.

1.1.1 Course Plan and Execution

This course will focus on foundations of machine learning. (Though applications are hinted throughout the course, applications are not the focus.) This is not an advanced course. This builds on top of the basic math knowledge a good undergraduate student might have obtained in the initial years of study. Though this course may expect the students to do some implementations, this course is not a programming/implementation/system building oriented course.

1.1.1.1 Major Topics/Modules

- Mathematical Foundations of Machine Learning
- Linear Methods in Machine Learning
- Neural Networks and Introduction to Deep Learning
- Selected topics Eg. (i) SVMs (ii) Decision Trees and Forests (iii) Ensembling (iv) Clustering and Unsupervised Learning

1.1.1.2 Delivery

Two lectures in a week. (> 20 Lectures) each of 1.5 Hrs. Come to the class 5 mins before time. Come prepared. In addition, a Tutorial (not all weeks) and Office Hours (announced on moodle), every week. It is expected that student go beyond the lectures and read/think and some extra work.

1.1.1.3 Evaluation

Course expects students to regularly work on the course, understand the basics and apply the concepts to solve problems. Final grade will be based on:

- Two short quizzes in the scheduled time for 15 % (7+8)
- A mid semester exam in the scheduled slot for 15 %
- A final exam in the scheduled slot for 25 %
- Regular home works (say around 3 problems (could vary between 2-4)) loosely associated with each lecture. These problems need to be solved regularly. The expected solutions are mostly handwritten, though there could also be some simple programming/scripting/visualizing based question. Relative weight: 25 to 30%. To take care of personal emergencies and busy schedules, students are given option to submit a lesser number of questions or select the best sub-set of answers. Typically this will be 60 out of 75 questions.
- Three assignments or mini projects worth 15 to 20%. These are well defined larger problems, that demand more work and go to multiple weeks.
- Minor changes in the course evaluation (5%) based on the practical changes/adjustments need to make.

1.1.2 Pre-requisites

This course assumes some background in mathematics, specially (i) Linear Algebra, (ii) Probability and Statistics, and (iii) Differential calculus. Background from undergraduate mathematics courses is good enough. Also some familiarity with programming, specially programming using python is expected.

1.1.3 How to get best out of this course?

It is assumed that the students taking this course are genuinely interested in the subject. To get best out of this course, the following are recommended:

- **Be Regular:** Be regular to the class/lecture. Reach classroom 5 mins before the start. Remain fully attentive. Engage in the discussions. Don't shy away from asking questions. Solve/Attempt home works immediately after the class, without delaying to the last minute. Use office hours and interactions with TAs for clearing doubts and getting personalized supports.
- **Practice Makes Perfect:** Many problems (home works or that you find in text books) may look to be simple. Good to still tryout and improve by practicing. If you notice that you are taking more time to solve the problems, your problem solving skills can be sharpened by practicing.
- **Scripting and Rapid Prototyping:** There could be many simple programming style problems. Do not treat them as an elaborate "software development process". In most cases, the objective is to use a computer to solve a sufficiently large problem than teaching you programming or program design. Also expectation is to teach you how to look and interpret data/learning. Python has emerged as the default programming language in this space, given its advantages in rapid prototyping, wide support packages/libraries, and compatibility with wide varity of hardware/software infrastructure.
- **Improve your visualization and imagination skills** You may want to visualize what happens when a mathematical operation takes place. This helps in understanding math in an intuitive manner, specially for machine learning tasks. Geometry is key to this.
- **Be comfortable in thinking beyond** What is taught in the lecture needs to be interpreted and expanded in your mind. Do not look at lecture notes as fact-list for your examination preparation. Ask basic questions like why and why not with out fear. May be in the entire process, you will find a novel algorithm in your name.!!
- **Read and Explore:** You are expected to read and explore beyond attending the lectures. However, this area is very rich in content on Internet. A student may find it impossible to understand everything related available on Internet. Don't worry. No hurry. Suggested mode: (i) Attend lectures, take simple notes (ii) Read lecture notes (if available!!) on the same day of the lectures itself (iii) Read/View text books/writeups/videos. (iv) Solve home works (v) go wider and wilder reading.
- **Learn from your classmates** You learn a lot from the classmates, and from discussions. However, maintain strict academic integrity while solving home works.

1.2 Problem Space

Let us start with our basic problem. We are given a number of examples in the form $\{\mathbf{x_i}, y_i)\}$ for $i = 1, \ldots, N$. For the simplicity, we assume that $\mathbf{x_i}$ is a real d-dimensional vector. And y_i is a scalar (say real number or an integer). Our interest is in finding a function $f()$ such thar $f(\mathbf{x_i})$ is same (or very similar) as y_i.

(At this moment, we will deal with the requirement of the "very similar" to be as close as possible for all the N training samples that we have. Later we should also make sure that the same function will work for all the samples that we come across in the future also (i.e., unseen samples).)

For example, $\mathbf{x_i}$ could be an email that is represented with a real vector. and y_i could be 0 or 1 corresponding to "spam" (1) or not (0). In this case it is a classification problem. One may also formulate the problem as classification of the email into "spam" (0), "personal" (1) and "professional" (2). In this case, this is a multiclass classification problem. You may also predict "how important/urgent" is an email by looking at the content. In this case, then y_i is a real number (say in the raneg $[0-1]$ where 0 means least urgent and 1 means extremely urgent.

1.2.1 Classification

In classification problems y_i is an integer. What value we assign is of not much significance. For example, some people use 0 and 1, while some where else we use -1 and $+1$ as the class IDs. The choice is often based on some conveniences (simpler form of equations!!). In both these cases, it is a "binary" classification. In many cases we also call these classes as ω_1 and ω_2.

Multi class classification where the number of classes is more than 2 is a popular case. Though multiclass classification is very popular and important in practice, many discussions in the linear classifiers assume that the number of classes is only 2. That makes the life simple.

1.2.2 Regression

In the case of regression, y_i is real quantity. It could be a real vector or a simple real number. Let us assume it as a real number at this stage.

1.3 Vectors and Matrices - I

We started with discussions on classifying emails. (or we could separate apples from oranges). How do we represent these physical entities? There are two popular ways in which we represent physical entities. Vectors in d dimensions. Sequences of Vectors/Observations. This makes the vectors fundamental to this area. These observations get transformed into knowledge with a set of transformations. Matrices (or sometimes Tensors) play a critical role in this. We are often interested in entities like vectors and matrices in linear algebra. They play a critical role in this course.

Vector A vector $\mathbf{x} \in R^n$ or $[x_1, x_2, \ldots, x_n]^T$. We think vectors as points in a space, with each element giving coordinate along an axis.

For example a fruit is represented in 2 dimension with colour and shape as dimension. Often these two measurements are real numbers.

Matrix A matrix $\mathbf{A} = [A_{ij}]$ of order $M \times N$ has MN elements. When $M = N$, it is a square matrix.

Types of Matrices You may also know a number of special matrices like (i) Diagonal matrix (ii) Triangular Matrix (iii) Singular Matrix (iv) More (??).

Linear Transformations Tools and techniques from Linear algebra provides a way of compactly and conveniently representing and manipulating sets of linear equations. For example,

$$a_1x_1 + a_2x_2 + \ldots + a_Nx_N = b$$

is a linear equation and can be written compactly as

$$\mathbf{a}^T\mathbf{x} = b.$$

In this case, $\mathbf{x}$ is transformed into a scalar b with the help of $\mathbf{a}$.

We can also linearly transform $\mathbf{x}$ by a matrix $\mathbf{A}$ as

$$\mathbf{y} = \mathbf{A}\mathbf{x}$$

A popular problem that we are interested (for many later lectures) is in finding *what is the best transformation* for our problem.

1.3.1 Operations on Matrices and Vectors

- **Dot product, inner product or scar product** of two vectors, $\mathbf{x}$ and $\mathbf{y}$ is often represented as $\mathbf{x} \cdot \mathbf{y}$ or $\mathbf{x}^T\mathbf{y}$ (more generally also as $< \mathbf{x}, \mathbf{y} >$. The product results in a scalar.
- $\mathbf{B} = \mathbf{x}\mathbf{y}^T$ is a matrix of size $d \times d$, if both $\mathbf{x}$ and $\mathbf{y}$ are $d \times 1$. *In this case,* $\mathbf{B}$ *is of rank 1. It has only one non-zero eigen value. Why?*
- $\mathbf{z} = \mathbf{A}\mathbf{x}$ is a vector. $\mathbf{A}$ has d columns. If $\mathbf{A}$ has $m(< d)$ number of rows, Then $\mathbf{A}$ is doing a *dimensionality reduction.*
- $\mathbf{A} + \mathbf{B}$ is feasible if both have the same size, and the result is a matrix.
- The multiplication of two matrices, $\mathbf{A}$ $\mathbf{B}$ is feasible when the number of columns of $\mathbf{A}$ is same as the number of rows of $\mathbf{B}$.
- Given a scalar α, we can compute the vectors αx and $\alpha\mathbf{A}$, by multiplying each element by α.

1.3.2 Norms

Norm A norm of a vector is informally the length of the vector, represented as $||\mathbf{x}||$. A popular way to visualize is as the Euclidean L2 norm as

$$\sqrt{\sum_{i=1}^{n} x_i^2}$$

A popular class of norms of our interest is Lp norm defined as:

$$||\mathbf{x}||_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{\frac{1}{p}}$$

When we $p = 2$, it is L2 norm, which we already saw. Another popular norm is $L1$ norm. $||\mathbf{x}||_1 = \sum_{i=1}^{n} |x_i|$

There are two special (pseudo) norms of interest

$$L_\infty = \max_i |x_i| \text{ and } L_0 = \text{ number of no-zero } x_i$$

There are also matrix norms. The most popular one is Frobenius norm defines as

$$||\mathbf{A}||_F = \sqrt{\sum_{i=1}^{M}\sum_{j=1}^{N} A_{ij}^2} = \sqrt{Tr(A^TA)}$$

1.3.3 Notes

Some of the key words that you need to know include: (i) Scalars, Vectors, Matrices and Tensors (ii) Multiplying Matrices and Vectors (iii) Identity and Inverse Matrices (iv) Linear Dependence and Span (v) Norms (vi) Special kinds of matrices and vectors (viii) Eigen decomposition and Singular value decomposition (ix) The determinant This notes may not discuss all these in detail. You may want to read any popular text book for refreshing the details.

1.4 Geometric Interpretations

Consider a set of points $\{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_N}\}$, each $\mathbf{x_i} \in R^2$. We can visualize this as a set of points on a 2D plane as in Figure **??**(a). The respective dimensions are the first and second dimensions of the point/sample $\mathbf{x}$. If $\mathbf{x}$ is an fruit, the first and second dimension could be the "colour" and "smell" of it. (Indeed, colour and smell will have to be captured with an appropriate sensor.) In practice, these sensor measurements are also scaled or normalized to certain ranges like $[-1, +1]$, $[0, 1]$ or $[0, 100]$.

Why do we have to normalize the data/vectors/samples?

Lines, Planes and Hyperplanes Consider the 2D plane example, all equations of the form

$$w_1 x_1 + w_2 x_2 = \mathbf{w}^T \mathbf{x} = 0$$

are lines in this plane.

When the data is in 3D, these are planes and when the data is more than 3 dimension, these are hyperplanes. You should be comfortable in imagining such geometric entities in high dimensionm. Most of the figures, we use, will be in 2D. With some effort, we can also draw 3D.

The equation of line we discussed above always pass through the origin. We will see how to relax this in the next lecture (Hint: *bias*)?

Sides of Planes/Lines When a sample $\mathbf{x} = [x_1, x_2]^T$ is on this line, then $\mathbf{w}^T\mathbf{x}$ is zero. All the points where this product is Positive is one side of the line and all the points where this dot product is negative is on the other side of the line.

In general the hyperplane divides the space into two half spaces $H1$ and $H2$ as $H1 = \{\mathbf{x}|\mathbf{w}^T\mathbf{x} > 0\}$ and $H2 = \{\mathbf{x}|\mathbf{w}^T\mathbf{x} < 0\}$. Don't worry about the equality case too much.

Angle between vectors We know that $\mathbf{x}^T\mathbf{y} = ||\mathbf{x}|| \cdot ||\mathbf{y}|| \cdot \cos\theta$. Where θ is the angle between the vectors.

$$\cos\theta = \frac{\mathbf{x}^T\mathbf{y}}{||\mathbf{x}|| \cdot ||\mathbf{y}||}$$

In many cases, both the vectors of interest could be "normalized" or unit norm, and the cosine angle between these vectors is just the dot product of the two vectors (since the denominator is one).

The cosine of the angle between two vectors is often used as a measure of similarity between two vectors. This quantity is in the range [-1,+1]. When the similarity is maximum, it is +1.

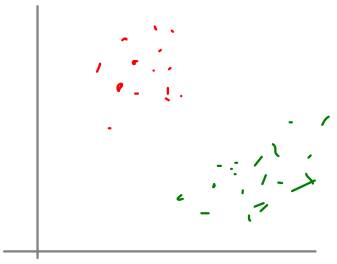


Figure 1.1: TBD

1.5 Nearest Neighbour Algorithm

Let us now look at a simple, yet practical, algorithm. We start by looking at a 2D visualization of emails in Figure **??**. The red dots are spam emails and green dots are non-spam emails.

Problem: We are given now, three new emails $\mathbf{x}_2$, $\mathbf{x}_2$ and $\mathbf{x}_3$. We have been asked, whether these are spam emails or not?.

It may be obvious for us that $\mathbf{x}_1$ is a spam email and $\mathbf{x}_2$ is a non-spam email. Why? We have a simple argument. All the emails similar to $\mathbf{x}_1$ are spams and we also treat $\mathbf{x}_1$ as spam. Similarly all emails similar to $\mathbf{x}_2$ are non-spam and we treat $\mathbf{x}_2$ as non-spam. Why is it? Two very similar emails will map to very similar points in the feature space. We can also use the distance between the points as similarity between the points. The smaller the distance, the larger the similarity.

How do we convert this into a computational algorithm? We can do it in two steps. (i) Given a test sample, find the nearest neighbour in the training data (ii) Assign the label of the neatest neighbour to the test sample.

This nearest neighbour classification algorithm works well in practice. However, to make it more robust to noisy data, we use K nearest neighbours instead of 1 nearest neighbour. As a result, label of the test sample is the majority of the labels in the neighbouring examples.

This leaves an open question. What is the value of K? Often it is a small odd number such as 3, 5 or 7.

- Why should K be odd?
- Can there be any time ties in the majority labels? How do we handle the tie breaks?
- What are the practical issues in implementing this algorithm for spam filters on gmail?

1.5.1 Distance Functions

In the previous section, we assumed that the distances are Euclidean. It need not be. There are many other possible distance functions. Nearest neighbour algorithm can use any distance function.

Metric Distance A distance function $d(\cdot,\cdot)$ is a metric if it obeys triangular inequality.

For points $\mathbf{x}$ $\mathbf{y}$ and $\mathbf{z}$, triangular inequality is defined as

$$d(\mathbf{x},\mathbf{y}) \leq d(\mathbf{x},\mathbf{y}) + d(\mathbf{y},\mathbf{z})$$

1.6 A note on notations

You may want to develop the skill of interpreting the notations without being explicitly written in all cases. In general small bold letters are vectors and capital bold letters are Matrices. (You will also see some Greek letters later.)

Suffixes like A_{ij} is the (i,j) th element of the matrix. $\mathbf{A_i}$ is the i th matrix and $\mathbf{x}_i$ is the ith vector. Transpose is represented as $*^T$. $\mathbf{x}^T$ is the row vector if $\mathbf{x}$ is a column vector. $\mathbf{A}^T$ is $N \times M$, if $\mathbf{A}$ is $M \times N$.

In machine learning, we are often interested in learning a set of coefficients. Frequently, they are represented as $\mathbf{w}$ or matrix $\mathbf{W}$. (However, not all learnable parameters are w.)

There are also many hyperparameters in the solution.

1.7 Home Works

1. We have a binary classification problem with 10 samples, 5 each from class A and B. All samples are in 2D. The data is given below with first 5 from class A and the next 5 from class B.

   $\mathcal{D} = \{[0.5, 0.5]^T, [0.75, 0.25]^T, [0.25, 0.75]^T, [1, 1]^T, [2, 2]^T,$

   $[-2, -2]^T, [-0.25, -0.25]^T, [-1.5, -2.5]^T, [-3, -2]^T, [-2, -3]^T\}$

   Given an unknown sample $\mathbf{q} = [0, 0]^T$, find its label using a simple K nearest neighbour algorithm with K = 1 and K = 3.

   A hand drawn sketch with approximate space is also expected.

2. Create a data set (say an xl sheet !) of the following data for all the players in India's WC Cricket squad 2019 (15 players).

   - Name
   - Age
   - Height of the player
   - Role (batsman=1, Bowler = 2, Wicket Keeper = 3, Al-rounder = 4)
   - Batting average (ODI)
   - Bowling Average (ODI)
   - Number of matches played (ODI).

   (a) Using a nearest neighbour scheme find who is the most similar player to Kumar Sangakara in the Indian team? Is it MS Dhoni? (Note thar Sangakara has also to be represented in the same feature space.)

   (b) Who is most similar to David Warner in Indian team? Is it Rohit Sharma?

   (c) Test one more players of your choice.

   May be that the results of Nearest Neighbour method is not meaningful here. What can we do to improve the similarity? Can you define a weighted distance function that solves this immediate worry?

   Submit data, results and analysis in handwritten form.

3. We know that the vector/sample $\mathbf{x}$ is on left of the hyperplane if $\mathbf{w}^T\mathbf{x} < 0$ and right (other side) if $\mathbf{w}^T\mathbf{x} > 0$.

   - Given $\mathbf{x_1} = [1, 2, 3]^T, \mathbf{x_2} = [0, 3, 4]^T, \mathbf{x_3} = [2, 4, 4]^T$, find 'a' hyper-plane $\mathbf{w}$ such that $\mathbf{w}^T\mathbf{x}_1 < 0$ and $\mathbf{w}^T\mathbf{x}_2 > 0$ $\mathbf{w}^T\mathbf{x}_3 > 0$
   - Find four points $\mathbf{x_1}$, $\mathbf{x_2}$, $\mathbf{x_3}$ and $\mathbf{x_4}$ in 4 dimension (d=4) such that that there exists no $\mathbf{w}$ that can keep $\mathbf{x}_1$ and $\mathbf{x}_2$ on one side and $\mathbf{x}_3$ and $\mathbf{x}_4$ on the other side. (similar to ExOR in 2D)

CSE 475: Statistical Methods in AI **Monsoon 2019**

SMAI-M-2019 2: Mathematical Foundations of ML - II

Lecturer: C. V. Jawahar *Date: 1 Aug 2019*

2.8 Problem Space -II

In the last lecture, we introduced the problem of separating an email as spam or non-spam. We also discussed a number of similar problems (like separating apples and oranges or a patient has certain disease or not) can all be abstracted into a classification problem. In general, our input is a vector in d dimension – $\mathbf{x}$. And our objective is to predict the y, an integer (classification) or a real number/vector (regression). Our objective is to learn a parameterized function $f(\mathbf{w}, \mathbf{x})$ that can predict y. A simple example is $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$.

How do we learn this function? That is the main story.

We are given many labeled examples $\mathcal{D} = \{(\mathbf{x_i}, \mathbf{y_i})\}$ where $i = 1, \ldots, N$ to learn the function (i.e., design the spam filter).

- **Training:** Training is the process of finding the function, (primarily finding the parameters $\mathbf{w}$), given sufficiently large training examples. This stage is mostly compute intensive. Outcome of the training is often the parameters/weights $\mathbf{w}$.
- **Testing:** Testing is the process of evaluating the function on a new sample (that is often unseen at the training time, like a new email that we receive) and predicting y. This is computationally light.
- **Performance Evaluation:** Given that we have now a spam filter, we would like to see how good is it objectively. A simple scheme could be how many emails are correctly classified. Percentage of the correctly classifier emails (a number in the range of 0-100) is a simple naive performance measure.

Q: Can you tell me why the percentage accuracy is a bad choice for a spam filter? What do we care in a spam filter?

There are a number of performance metrics useful for us. Here are some of them (i) Percentage Accuracy (ii) Hit Ratio (iii) Miss Ratio (iv) False Positive Rates (v) Precision (vi) Recall (vii) Average Precision (AP) (viii) AUC (Area under the curve) etc.

Q: Read and understand about these metrics on your own.

How do we learn the function in practice? What we have is only N labeled examples. This is the popular strategy in our labs/experiments. (In practice, there is a real world deployment and seeing many many more real unlabeled emails.). The standard protocol followed is like this. We split the samples into two sets "Train" and "Test". Or we split $\mathcal{D}$ into two mutually exclusive subsets $\mathcal{D}_1$ and $\mathcal{D}_2$.

In some cases, a well defined partitions exists, in some other cases, a random partition is enforced. (If no detail is available let us split as 80:20 for Train:Test).

To summarize, this is what we will do in the exercises:

- **Input:** We are given N labeled examples. we artificially split the data into two mutually exclusive sets train and test.
- **Training:** We make an assumption of the form of the function (eg. $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$) and we choose our favorite algorithm to learn the parameters of the function $\mathbf{w}$.
- **Testing:** We evaluate the solution on the test-set (say $\mathcal{D}_2$ and predict the output for each one independently.
- **Performance Evaluation:** We evaluate the performance of our solution by comparing the predicted output with actual (ground truth or what is provided to us) output.
  - Percentage accuracy is a simple popular measure for classification.
  - Mean square error (MSE) is a popular error measure for regression like problem.

Point of Concern The seperation of Train and Test set is very imnportant. It is quite likely that we may use test set for training, inadvertently. This has to be strictly avoided for practicing ML.

- For example, your code will not use the data but you (as a person) will look at the test data many time, you are indirectly using the test data for training. This happens in practice. This should be avoided.

– Sometime, we do not see the test-data as individuals, but as a community, we see it many manmy times (like popular data sets) leading/encouraging development of algorithms that use this information since it is rewarding to do well on such a specific public benchmark. This also leads to algorithms that do well on the train test but not well on *real novel situations*.

Problem: In the previous protocol, we assumed that each sample is tested independently. What about we test all the N_2 samples together and output a ranked list of samples based on the confidence. This is a popular requirement in many ranking style of problems. (eg. information retrieval). A popular measure in such cases, is "Average Precision". With one or two numerical examples, explain average precision.

2.9 Matrices and Vectors - II

2.9.1 Augmented Vectors and Feature Maps

We started with the problem of $\mathbf{w}^T\mathbf{x} >$ or < 0. Note that this assumes that our line/plane/hyperplane pass through origin and it does not give us a complete family of solutions. We should also add an additional w_0 to address this. Note that this is same as $\mathbf{w}^T\mathbf{x} >$ or $< w_0$

The notation could be simpler if we do not have w_0. We do that by augmenting the vector $\mathbf{x}$ with an additional quantities. i.e., the new $\mathbf{x}$ is the old $\mathbf{x}$ with an additional 1 concatenated at the end. Similarly the $\mathbf{w}$ is augmented with w_0 and the decision function gets simplified as

$$y = f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} \tag{2.1}$$

Note that we did not introduce a different notation with and without augmentation. This is to make the notations simpler. (Text books (such as Duda and Hart) may be using different notations for these). Hope you appreciate the convenience of augmenting with 1.

What more we can do with augmentation (or explicit feature maps)? We can infact create a new vector from the old ones. This also generalizes the trick of augmentation.

Consider that our original vector is $[x_1, x_2]^T$. We now know how to create a new vector $[x_1, x_2, 1]$. Why not $[x_1^2, x_2^2, x_1x_2, x_1, x_2, 1]$? Such a modification will allow us to learn a new model $\mathbf{w}^T\mathbf{x}$ as

$$w_5x_1^2 + w_4x_2^2 + w_3x_1x_2 + w_2x_1 + w_1x_2 + w_0 \tag{2.2}$$

which is really a quadratic function. Our linear algorithm (that we see soon) is able to learn nonlinear models too. The objective of introducing this at this stage is to convince that the algorithms that we will discuss are very powerful and useful. Linearity does not constrain us too much.

2.9.1.1 Properties/Terms

1. **Norm** A norm of a vector is informally the length of the vector, represented as $||\mathbf{x}||$. A popular way to visualize is as the Euclidean L2 norm as
$$\sqrt{\sum_{i=1}^{n} x_i^2}$$
2. L^p **Norms** A popular class of norms of our interest is Lp norm defined as:
$$||\mathbf{x}||_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{\frac{1}{p}}$$
When we $p = 2$, it is L2 norm, which we already saw. Another popular norm is $L1$ norm. $||\mathbf{x}||_1 = \sum_{i=1}^{n} |x_i|$
There are two special (pseudo) norms of interest
$$L_\infty = \max_i |x_i| \text{ and } L_0 = \text{ number of no-zero } x_i$$
There are also matrix norms. The most popular one is Frobenius norm defines as
$$||\mathbf{A}||_F = \sqrt{\sum_{i=1}^{M}\sum_{j=1}^{N} A_{ij}^2} = \sqrt{Tr(A^TA)}$$
3. **Distance Or Similarity functions** that can compare two vectors.
   (a) Euclidean Distance
   (b) Weighted Euclidean (L2) distance
   (c) Cosine Similarity
   (d) ?? (your own cricket distance. But is it metric?)

2.9.2 Matrices

It is assumed that you are familiar with the notion of matrices. Here are some terms that you should revise, practice on how to numerically compute, if appropriate.

1. **Symmetric, Identity, Triangular** matrices
2. **Rows, Columns, Row-space, column-space**

3. **Determinant**
4. **Rank**
5. **Trace**
6. **Notion of Linear Independence**
7. **PD, PSD**
8. **Graph-Schmidt Orthogonalization**

Many known results

- What is $(ABC)^T$?
- What is $(ABC)^{-1}$
- Is AB same as BA?

2.10 Interpretations

2.10.1 Vectors

- Vectors and Directions
- Geometric Interpretation of $\mathbf{W}$ in $\mathbf{w}^T\mathbf{x}$
- Basic Vectors, Span

2.10.2 Matrices

- Matrix Multiplication as Linear Transformation
- Solving $\mathbf{Ax} = \mathbf{b}$
- Solving $\mathbf{Ax} = \mathbf{0}$
- Disadvantages of closed form solution for solving equations
- Solving $\mathbf{Ax} = \mathbf{b}$ when $\mathbf{A}$ is non square or rank deficient. Some special cases.
- Translation, Rotation and Scaling of Data
- Normalization of data

2.11 Home Works (Submit by Aug 9 5pm

1. We know that the cosine similarity is a popular way to compare two vectors. Write the expression. This is in the range of [-1,+1]. When can it be -1, 0 and +1? Demonstrate with an example in 4 dimensional space.

2. Consider a set of samples $\mathcal{D} = \{\mathbf{x_i}, i \in 0, 1, 2, ..., n\}$. We also have a linear classifier defined by $\mathbf{w}$ that classifies samples based on whether $\mathbf{w}^T\mathbf{x_i}$ is greater than zero or less than / equals to zero into class A or B respectively.

   - If all samples are multiplied by a scalar $\alpha \in R^+$ as $\alpha\mathbf{x}_i$, will the accuracy of the classifier change on this set? Why?
   - If all samples are multiplied by independent random scalars α_i as $\alpha_i\mathbf{x}_i$, will the accuracy of the classifier remain unchanged? Why?
   - If the samples were linearly transformed by a matrix $\mathbf{A}$ ($\mathbf{Ax}$ that is orthonormal (i.e., $\mathbf{A^TA} = \mathbf{AA^T} = \mathbf{I}$), will the accuracy change?
   - If the samples were linearly transformed by a matrix $\mathbf{A}$ that is rank deficient (with determinant zero), will the accuracy remain unchanged?
   - From the above specific examples, discuss when and how the accuracy will remain unchanged, in general. Make your argument scientific and analytical as much as possible.

3. Consider a problem of classifying (testing) samples in 2D (i.e., $[x_1, x_2]^T$). Each sample is augmented with 1 to get $\mathbf{x} = [x_1, x_2, 1]^T$. Classification is done as "decide as class A if $\mathbf{w}^T\mathbf{x} > 0$ else decide as class B". Where $\mathbf{w} = [w_1, w_2, w_3]^T$.

   Write a program that generates 50 random samples from class A and another 50 random samples from class B leading to a total of 100 samples. For all the samples x_1 and x_2 are in the range [-1,+1]. (Indeed there is no structure to these classes, when we generate samples randomly. It is OK for today).

   (i) For the following $\mathbf{w}$ vectors calculate the accuracy (as percentage). (a) $[1, 1, 0]^T$ (b) $[-1, -1, 0]^T$ (c) $[0, 0.5, 0]^T$ (d) $[1, -1, 5]^T$ (e) $[1.0.1.0, 0.3]^T$. Report the accuracies. (ii) Provide the plots (data + decision boundary) for (a), (d) and (e). (iii) Explain why do you get these accuracies given that data is random with equal probabilities. Where you able to guess before the experiment/code?

CSE 475: Statistical Methods in AI **Monsoon 2019**

SMAI-M-2019 3: Mathematical Foundations of ML - III

Lecturer: C. V. Jawahar *Date: 5 Aug 2019*

3.12 Problem Space - III

3.12.1 How do we formulate the Training Problem?

Let us come back to our problem. We are given training examples $\{(\mathbf{x_i}, y_i)\}$ $i = 1, \ldots, N$. What do we want to do? We want to find the most appropriate $\mathbf{w}$ such that we can mimic y_i as $f(\mathbf{w}, \mathbf{x}_i)$.

Indeed we may not find a single $\mathbf{w}$ that can make $y_i = \mathbf{w}^T\mathbf{x_i}$ for all i. This could be due to various reasons including errors, noise or uncertainty in the data. Therefore we want to model the problem as find the "most appropriate" $\mathbf{w}$. This naturally lead to an optimization problem. Most appropriate in what sense? We need to define an appropriate sense or objective that can be computed. This is the objective function. In machine learning, we also use the term loss function or error function frequently. All these are used with very similar meanings.

Our problem is then to find $\mathbf{w}$ that minimizes

$$Total{-}Loss = \sum_{i=1}^{N} Loss{-}Per{-}Sample = \sum_{i=1}^{N} L(\mathbf{w}, \mathbf{x_i}, y_i)$$

Why do we have to sum over i? Why not products? That is also possible. We do summation so that differentiation is easier later. (do you remember how to differential $u + v$ and uv?). There may be many different ways in which you can define loss functions.

Q: Do you see any other advantage or disadvatage of products over sum? Which will be more sensitive to outliers? (samples which may have very larger error). Or samples where the loss vanishes (becomes zero)?

3.12.2 Loss Function

Consider the regression problem in 1D. You have (x_i, y_i). By augmenting 1, x_i becomes a 2 dimensional vector $\mathbf{x}_i$. Our problem is to find the vector $\mathbf{w} = [w_1, w_0]^T$ such that the model is $y = w_1x_1 + w_0$. Look at this as a line fitting.

Error or loss in this case is the difference between the model prediction and actual.

$$L(\mathbf{w}, \mathbf{x_i}, y_i) == (y_i - \mathbf{w}^T\mathbf{x_i})^2$$

We square the error such that no loss is negative. This is required since we will now be adding the loss from different examples to get the total loss.

And the total loss or objective is then sum over all the examples

$$J(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \mathbf{w}^T\mathbf{x_i})^2 \tag{3.3}$$

Note that we have only one $\mathbf{w}$ for all the samples.

Similarly for a classification problem, loss can be 1 if the classification is wrong and 0 if the classification is correct.

Let us assume that the classifier is

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w}^T\mathbf{x} > 0 \\ -1 & \text{else} \end{cases} \tag{3.4}$$

$$J(\mathbf{w}) = \frac{1}{N}\sum_{i=1}^{N}(1 - y_i \cdot f(\mathbf{w}, \mathbf{x_i})) \tag{3.5}$$

- Q: If we had used a $0 - 1$ convention for the classes, how the equation could have been written?
- Q: The problem with this loss is that this is not differentiable. Why?

3.12.3 Optimization

The optimization problem we need to solve is

$$minimze \;\; J(\mathbf{w}) \tag{3.6}$$

Though we know what problem to solve, very often we can not find the "best $\mathbf{w}$" in practice. There are two prominent classes of optimization problems:

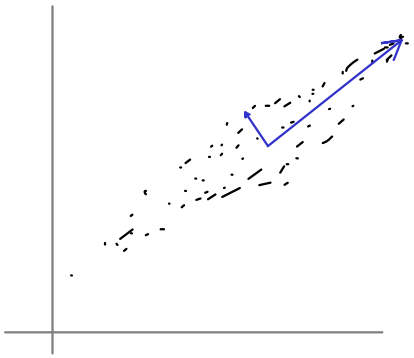
- Convex optimization. A well behaved class of problem. You can find the optima. And often efficiently.
- Non-Convex optimization problem. This is a class of nasty optimization problems. Unfortunately, we will encounter them very frequently. In this case, very often, we have to be happy with "a" minima/solution and not the best solution.

More details of these classes of problems is beyond the scope of this course.

3.13 Vectors and Matrices -III

1. When is $\mathbf{Ax} = \mathbf{b}$ uniquely solvable, when $\mathbf{A}$ is a square matrix?
2. When $\mathbf{A}$ has more rows than column or when $\mathbf{A}$ has more columns than rows, how do we solve, and what type of a solution we obtain?

3.13.1 Tensors



Eigen vectors.

TODO

PD:

3.14 Eigen Values and Eigen Vectors

Eigen values and eigen vectors of a square matrix A are defined by the characteristic equation

$$\mathbf{Ax} = \lambda\mathbf{x}$$

Here $\mathbf{x}$ is the eigen vector and λ is the eigen value.

- Q: How do we compute eigen values and eigen vectors with hand for small matrices?
- Q: How many non-zero eigen values could be there for $\mathbf{A}$ of $n \times n$? n
- Q: How are λ_i related to determinant and trace?
- Q: Can eigen values be imaginary? When? yes
- Q: What can you say about the PD and PSD from eigen values?

3.14.1 Eigen Decomposition of a matrix

Let $\lambda_1, \ldots, \lambda_n$ be the eigen values and $\mathbf{v_1}, \ldots, \mathbf{v_n}$ be the eigen vectors of the $n \times n$ matrix $\mathbf{A}$

We know that

$$\mathbf{A}\mathbf{v}_i = \lambda_i \mathbf{v}_i \quad \forall i$$

Let $\mathbf{V}$ be a $n \times n$ matrix with columns as $\mathbf{v_1}, \ldots, \mathbf{v_n}$.

$V = [v_1\ v_2\ \ldots v_n]$

$$\mathbf{AV} = \mathbf{\Lambda V} = \mathbf{V\Lambda}$$

where Λ is the diagonal matrix of eigen values.

$$A = \mathbf{V\Lambda V^T}$$

since eigen vectors are orthogonal. We can also write it as

$$\mathbf{A} = \sum_{i=1}^{n} \lambda_i \mathbf{v}_i \mathbf{v}_i^T$$

We will see how the eigen decomposition of a square matrix A is related to the SVD of $\mathbf{A}$. And also in the case of non square matrices, how the SVD of AA^T and A^TA are related to the eigen values and eigen vectors of A.

Eigen Spectrum

It is good to observe the eigen values in a sorted manner as a spectrum to observe t

3.13 Vectors and Matrices -III

1. When is $\mathbf{Ax} = \mathbf{b}$ uniquely solvable, when $\mathbf{A}$ is a square matrix?
2. When $\mathbf{A}$ has more rows than column or when $\mathbf{A}$ has more columns than rows, how do we solve, and what type of a solution we obtain?

3.13.1 Tensors

3.14 Eigen Values and Eigen Vectors

Eigen values and eigen vectors of a square matrix A are defined by the characteristic equation

$$\mathbf{Ax} = \lambda \mathbf{x}$$

Here $\mathbf{x}$ is the eigen vector and λ is the eigen value.

- Q: How do we compute eigen values and eigen vectors with hand for small matrices?
- Q: How many non-zero eigen values could be there for $\mathbf{A}$ of $n \times n$?
- Q: How are λ_i related to determinant and trace?
- Q: Can eigen values be imaginary? When?
- Q: What can you say about the PD and PSD from eigen values?

3.14.1 Eigen Decomposition of a matrix

Let $\lambda_1, \ldots, \lambda_n$ be the eigen values and $\mathbf{v}_1, \ldots, \mathbf{v}_n$ be the eigen vectors of the $n \times n$ matrix $\mathbf{A}$.

We know that

$$\mathbf{A}\mathbf{v}_i = \lambda_i \mathbf{v}_i \quad \forall i$$

Let $\mathbf{V}$ be a $n \times n$ matrix with columns as $\mathbf{v}_1, \ldots, \mathbf{v}_n$.

$$\mathbf{AV} = \mathbf{AV} = \mathbf{VA}$$

where A is the diagonal matrix of eigen values.

$$A = \mathbf{v}A\mathbf{v}^T$$

since eigen vectors are orthogonal. We can also write it as

$$\mathbf{A} = \sum_{i=1}^{n} \lambda_i \mathbf{v}_i \mathbf{v}_i^T$$

We will see how the eigen decomposition of a square matrix A is related to the SVD of A. And also in the case of non square matrices, how the SVD of AA^T and A^TA are related to the eigen values and eigen vectors of A.

Eigen Spectrum

It is good to observe the eigen values in a sorted manner as a spectrum to observe it

3.15 Singular Value Decomposition (SVD)

Singular value decomposition is a very popular factorization scheme with many applications in machine learning. The singular value decomposition (SVD) is a factorization of a real or complex matrix. It has many useful applications in signal processing, statistics, machine learning and optimization. In general, the matrices that we deal with are real.

Formally, the singular value decomposition (SVD) of an $m \times n$ matrix $\mathbf{A}$ is a factorization of the form

$$\mathbf{A = UDV^T}$$

3.15.1 Minor Variation in Notations

You see minor variations in the notation of SVD across resources. This comes from whether your original matrix $\mathbf{A}$ has more rows or columns. If $\mathbf{A}$ is of $m \times n$, the rank of $\mathbf{A}$ can be $\min(m, n)$. In many applications of machine learning, we see that the rank of $\mathbf{A}$ is often much smaller than $\min(m, n)$. This is because data has a structure. And ML techniques thrive in situations where there is some structure for the data.

- U is $m \times m$, $U^T U = I_m$, D is diagonal $m \times n$ and V is $n \times n$ as $V^T V = VV^T = I_n$.

  One may look at U as an $m \times m$ real or complex unitary matrix, D is an $m \times n$ rectangular diagonal matrix with non-negative real numbers on the diagonal, and V^T (the conjugate transpose of V, or simply the transpose of V if V is real) is a $n \times n$ unitary matrix.

- Alternatively, U is $m \times n$, $U^T U = I_n$, D is diagonal $n \times n$ and V is $V^T V = VV^T = I_n$.

In both casesm **UD** is finally the same. Is it so?

Q: Convince that these two different ways will not lead to different interpretations.

Notes

- The diagonal entries D_{ii} of D are known as the singular values of M. If D is interpreted as $m \times n$ with $m < n$, then the last $m - n$ rows are just zeros
- The m columns of $\mathbf{U}$ and the n columns of $\mathbf{V}$ are called the left-singular vectors and right-singular vectors of $\mathbf{A}$, respectively.
- Note that $U^T U = I$ and $V^T V = VV^T = I$ The singular value decomposition and the eigen decomposition are closely related. We know that the eigen decomposition leads to

$$\mathbf{A} = \sum_{i=1}^{N} \lambda \mathbf{x}\mathbf{x}^T$$

Do you see the connection between this and the SVD?

3.15.2 Interpretations

- The left-singular vectors of M are eigenvectors of MM^T.
- The right-singular vectors of M are eigenvectors of $M^T M$.
- The non-zero singular values of M (found on the diagonal entries of D) are the square roots of the non-zero eigenvalues of both $M^T M$ and MM^T.

These are quite simple to see. For example if we look $\mathbf{A}^T\mathbf{A}$ as

$$\mathbf{A}^T\mathbf{A} = (UDV^T)^T(UDV^T) = VDU^TUDV^T = VD^2V^T$$

3.15.3 Examples of Effective Uses

Applications that employ the SVD include computing the pseudoinverse, least squares fitting of data, matrix approximation, and determining the rank, range and null space of a matrix. Let us assume $A = UDV^T$ Or

$$\mathbf{A} = \sum_i D_i u_i v_i^T$$

$$A^{-1} = (UDV^T)^{-1} = VD^{-1}U^T$$

D^{-1} is easy to calculate since it is diagonal in n flops. If A is singular, one can find the approximation by discarding the zero elements. $D_i^{-1} = \frac{1}{D_i}$ is $D_i > t$ and zero otherwise. One can solve $Ax = b$ with the help of this inverse.

3.15.4 Other Factorization Methods

SVD is not the only matrix decomposition technqiues that we use. There are many more. Urge you to be also familiar with other such as

- Cholesy
- QR, LU etc.

SVD:

$A = UDV^T$

$A = (m \times n)$

$U^TU = I_m$

$VV^T = I_n$

D = diag

V^T: unitary.

$A = \sum_{i=1}^{N} \lambda x x^T$

Do you see the connection between this and the SVD?

3.15.2 Interpretations

- The left-singular vectors of M are eigenvectors of MM^T.
- The right-singular vectors of M are eigenvectors of M^TM.
- The non-zero singular values of M (found on the diagonal entries of D) are the square roots of the non-zero eigenvalues of both M^TM and MM^T.

These are quite simple to see. For example if we look $\mathbf{A}^T\mathbf{A}$ as

$$\mathbf{A}^T\mathbf{A} = (UDV^T)^T(UDV^T) = VDU^TUDV^T = VD^2V^T$$

3.15.3 Examples of Effective Uses

Applications that employ the SVD include computing the pseudoinverse, least squares fitting of data, matrix approximation, and determining the rank, range and null space of a matrix. Let us assume $A = UDV^T$ Or

$$\mathbf{A} = \sum_i D_i u_i v_i^T$$

$$A^{-1} = (UDV^T)^{-1} = VD^{-1}U^T$$

D^{-1} is easy to calculate since it is diagonal in n flops. If A is singular, one can find the approximation by discarding the zero elements. $D_i^{-1} = \frac{1}{D_i}$ is $D_i > t$ and zero otherwise. One can solve $Ax = b$ with the help of this inverse.

3.15.4 Other Factorization Methods

SVD is not the only matrix decomposition techniques ... Urge you to be also familiar with other such as

- Cholesky
- QR, LU etc.

Notes

- Singular value decomposition is a very popular factorization scheme with many applications in machine learning. The singular value decomposition (SVD) is a factorization of a real or complex matrix. It has many useful applications in signal processing, statistics, machine learning and optimization. In general, the matrices that we deal with are real.

  Formally, the singular value decomposition (SVD) of an $m \times n$ matrix $\mathbf{A}$ is a factorization of the form

  $$\mathbf{A} = \mathbf{UDV^T}$$

- **Minor Variations in Notations**

  You see minor variations in the notation of SVD across resources. This comes from whether your original matrix $\mathbf{A}$ has more rows or columns. If $\mathbf{A}$ is of $m \times n$, the rank of $\mathbf{A}$ can be $\min(m, n)$. In many applications of machine learning, we see that the rank of $\mathbf{A}$ is often much smaller than $\min(m, n)$. This is because data has a structure. And ML techniques thrive in situations where there is some structure for the data.

  - U is $m \times m$, $U^TU = I_m$, D is diagonal $m \times n$ and V is $n \times n$ as $V^TV = VV^T = I_n$.

    One may look at U as an $m \times m$ real or complex unitary matrix, D is an $m \times n$ rectangular diagonal matrix with non-negative real numbers on the diagonal, and V^T (the conjugate transpose of V, or simply the transpose of V if V is real) is a $n \times n$ unitary matrix.

  - Alternatively, U is $m \times n$, $U^TU = I_n$, D is diagonal $n \times n$ and V is $V^TV = VV^T = I_n$.

  In both casesm **UD** is finally the same. Is it so?

  Q: Convince that these two different ways will not lead to different interpretations.

  $D_{i,i}$ = singular value of M

- The m columns of $\mathbf{U}$ and the n columns of $\mathbf{V}$ are called the left-singular vectors and right-singular vectors of $\mathbf{A}$, respectively.
- Note that $U^TU = I$ and $V^TV = VV^T = I$ The singular value decomposition and the eigen decomposition are closely related. We know that the eigen

3.16 Interpretations

Example 1 Consider a vector $\mathbf{x} \in R^d$. Let us now compute a $d \times d$ square matrix $\mathbf{A} = \mathbf{x}\mathbf{x}^T$.

We know that the matrix $\mathbf{A}$ is $d \times d$. However, what is the rank of $\mathbf{A}$? (It can be at max r. But that is a useless answer for this problem.) It is 1. Why? Each row of $\mathbf{A}$ is only a linear multiple of the other row. first row is $x_1\mathbf{x}$. Second row is $x_2\mathbf{x}$ and so on. Or we can express a row as a multiple of another one. And there fore there is only one linearly independent row.

Q: What about columns? How many linearly independent columns be there? Can you argue?

Q: If we take SVD of $\mathbf{A}$, what do we obtain U, D and V? If we compute eigen values and eigen vectors of $\mathbf{A}$, what do we obtain? Validate your answers analytically and then also verify it with a numerical library.

Example 2: Data matrix Consider a phenomenon/process that gives us data on a straight line. i.e., $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_N}$. We now create a matrix $\mathbf{D}$ by arranging these vectors as columns. i.e., $\mathbf{D}$ is a $2 \times N$ matrix. What is the rank of $\mathbf{D}$?

If all columns of $\mathbf{D}$ are samples from on a line we can easily see that any column can be obtained from another column with a scalar multiplication. Thus $\mathbf{D}$ is rank 1.

Note: In general **Data matrix D** is obtained by keeping samples as columns. Such matrices are of $d \times N$ in size. Which one is higher in practice, d or N?

Q: What about samples that lie on a plane in 3D? What should be the rank of this? What about hyperplanes in d dimension?

Q: Can you guess (and later empirically validate) what should be the SVD of $\mathbf{D}$?

Example 3: Noisy Data Matrix In real life, the situation is not that ideal, even if the process is a simple linear model, you may not get samples that can lie exactly on a line/plane/hyperplane. This is what often we see in data sets.

In such cases, the rank or non-zero eigen/singular values do not really provide insight. For example, an eigen value of very small magnitude may be practically as good as zero. This lead to "discarding" eigen/singular values that are very small (compared to the major ones).

For example, consider our $\mathbf{D}$ of $2 \times N$. If we have one sample not on the line, (while all other $N-1$ are on). What do you see in SVD? You get a major eigen value/singular value and then the second one that is non-zero is very small. A popular question is can we then find a "rank 1" approximation of $\mathbf{D}$ and remove the influence of this "odd" sample. Yes, indeed. We can take SVD of $\mathbf{D}$ as UDV^T. Then make the seond singular value zero. Then compute back UDV^T.

(Note: Don't get confused with the data matrix D and SVD's D).

Q: Can you visualize what happens to this on data with a small python code?

3.17 Subspaces

In general, it is observed that though the data is d dimensional (when you observe, curate), it lies in a lower dimension for all practical purposes. This is a sub-space of the original space.

- Lines in 2D
- Planes in 3D
- Lines in 3D
- Planes in 4D.
- d' dimensional hyperplanes in d dimensional space.

A line is 3D can be understood as all points that are in certain direction from a fixed point. Or

$$\{\mathbf{x}|\mathbf{x} = \mathbf{x}_0 + u\}.$$

We can define a subspace in this manner.

$$\{\mathbf{x}|\mathbf{x} = \mathbf{x}_0 + \sum_{i=1}^{d'} \mathbf{u}_i\}$$

There are only d' linearly independent vectors in this sub-space. (You will see more formal details in Chapter 2 (2.6?) of the book).

Let us come back and ask the question of why do data lie in a sub-space? Most physical processes generate data based on some simple model. However, when we are capturing the data, we have no cue about what is the inherent dimension of the data.

Think of the problem of email classification as spam or non-spam. As a designer of the algorithm, we land up defining and extracting a feature set based on the Vocabulary of English. (Q: What is the rough size of this?). However, in practice the data/problem is in a much low-dimension. Unfortunately, we have no systematic way to capture the data in the lower dimension. (Note: in the case, of emails, it is NOT 2. We may be showing examples in 2D in the lecture, does not mean that the real life problems are often in 2D!!).

3.18 Dimensionality Reduction

When the data lies in a lower-dimensional subspace, it is natural to ask the question, why are you solving in the original space and why not move to the lower dimensional space. It is a valid question, and popularly this is done by **dimensionality reduction**.

In general, during dimensionality reduction, we transform the data from high dimension (say d) to low dimension (say d'). In linear dimensionality reduction schemes, this is achieved by a matrix multiplication.

$$\mathbf{z} = \mathbf{A}\mathbf{x}$$

This is a linear dimensionality reduction.

We can also do nonlinear dimensionality reduction with nonlinear functions (such as Neural Networks) as

$$\mathbf{z} = f(\mathbf{w}, \mathbf{x}).$$

In either case, our objective is to find an effective lower dimensional space where data lie. Or find a space where the problem can be solved better.

3.18.1 Curse of Dimensionality

There are also many curse for the dimensionality. If we have a feature/data representation that is very large, it creates many practical problems for us. For example, if we are estimating some data statistics (like covariance matrices), the number of parameters to estimate increase quadratically with the number of dimensions. And indeed, this demand more and more samples to estimate these parameters.

There are many other practical issues like:

- Combinatorics
- Sampling
- Optimization
- Distance Function
- etc.

Urge to read articles such as Wiki on curse of dimensionality to appreciate this behavior.

3.18.2 Low-Dimensional Projections

It is common to create low-dimensional projections of large dimensional data for effective and efficient machine learning algorithms

3.19 Home works (Submit by 12 Aug 5pm)

1. With hand compute the eigen values and eigen vectors of the following matrix:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

   What is the trace, determinant and rank of this matrix? How is this related to the eigen values/vectors. Demonstrate.

2. Consider a linear transformation (such as a dimensionality reduction) that maps a q-dimensional vector $\mathbf{x}$ to a p dimensional vector $\mathbf{y}$ (where $p \leq q$; BTW, for dimensionality reduction, in practice, p can be much smaller than q.) as

$$\mathbf{y}_i = \mathbf{A}\mathbf{x}_i$$

   (a) What are the dimensions of the matrix $\mathbf{A}$?

   (b) Can there exists a matrix $\mathbf{A}$ such that the Euclidean distance between $\mathbf{y}_1$ and $\mathbf{y}_2$ is same as $\mathbf{x}_1$ and $\mathbf{x}_2$. Under what conditions on $\mathbf{A}$ and $\mathbf{x}$, is this feasible?

   (c) Create a specific example and demonstrate $\mathbf{A}$ when (a)$q = 2$ and $p = 2$, (b) $q = 2$ and $p = 1$ and (c) $q = 4$ and $p = 2$. If no examples like these are feasible, explain/prove why.

3. Consider a line in 2D (say line l as $w_1x_1+w_2x_2+w_3 = 0$)

   (a) Assume we have a set of N points on this line $\mathcal{D}_1 = \{\mathbf{x}_i = [x_1^i, x_2^i]^T\}$. Let the mean of these points be $\mu = [\mu_1, \mu_2]^T$. How many non-zero eigen values will be there for the following matrices?

$$\mathbf{A}' = [x_1^1 - \mu_1, x_2^1 - \mu_2][x_1^1 - \mu_1, x_2^1 - \mu_2]^T$$

$$\mathbf{A} = \frac{1}{N}\sum_{i=1}^{N}[x_1^i - \mu_1, x_2^i - \mu_2][x_1^1 - \mu_1, x_2^i - \mu_2]^T$$

   or compactly

$$\mathbf{A} = \frac{1}{N}\sum_{i=1}^{N}[\mathbf{x}_i - \mu][\mathbf{x}_i - \mu]^T$$

   (b) Assume we have a set of points on a line perpendicular to the line l and passing through μ. (What is the equation of this line?) Let $\mathcal{D}_2 = \{\mathbf{x}_j = [x_1^j, x_2^j]^T\}$ be a set of N points on

this perpendicular line. Let the mean of these points be $\mu = [\mu_1, \mu_2]^T$. How many non-zero eigen values will be there for the following matrices?

$$B' = [x_1^1 - \mu_1, x_2^1 - \mu_2][x_1^1 - \mu_1, x_2^1 - \mu_2]^T$$

$$B = \frac{1}{N}\sum_{i=1}^{N}[x_1^i - \mu_1, x_2^i - \mu_2][x_1^i - \mu_1, x_2^i - \mu_2]^T$$

or compactly

$$\mathbf{B} = \frac{1}{N}\sum_{i=1}^{N}[\mathbf{x}_i - \mu][\mathbf{x}_i - \mu]^T$$

(c) Now consider the main question. Consider a set of points "around" the line with mean as μ. Let us compute the covariance matrix as:

$$\Sigma = \frac{1}{N}\sum_{i=1}^{N}[\mathbf{x}_i - \mu][\mathbf{x}_i - \mu]^T$$

What will be the eigen value and eigen vector of the Σ. Discuss.

(d) Write a python program to create 1000 points around the line (not just on the line). Compute covariance matrix, eigen values and eigen vectors. (i) Plot the data (ii) Plot the eigen vectors (first with red and second with green starting from mean. (iii) Overlay these two plots and create a single plot.

Note: this question is long. But the answer is not that. Conceptual understanding of this question could be the key for appreciating PCA in one of the next lectures.

CSE 475: Statistical Methods in AI **Monsoon 2019**

SMAI-M-2019 4: Mathematical Foundations of ML - IV

Lecturer: C. V. Jawahar *Date: 8 Aug 2019*

4.20 Problem Space - IV

In the last lecture we looked at the learning problem and

- Understood it as an optimization problem of an appropriate loss/objective function.
- We also defined the data/examples $\mathcal{D}$ into two subsets $\mathcal{D}_{Tr}$ and $\mathcal{D}_{Te}$ as the subsets used for "Training" and "Testing"

Given this background, let us ask a critical question? What are we optimizing over?

- $\mathcal{D}_{Tr}$ or $\mathcal{D}_{Te}$ or $\mathcal{D}$?.

Since our objective is to define a computational procedure, we work only on $\mathcal{D}_{Tr}$.

This means our problem is something like:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_{i \in \mathcal{D}_{Tr}} ||f(\mathbf{x}_i, \mathbf{w}) \leftrightarrow y_i||$$

Where $\leftrightarrow$ compares the predictions (i.e., $f(\mathbf{x}_i, \mathbf{w})$) with that of "truth" (i.e., y_i). It could be something as simple as difference.

The above formulation seems to be correct (and that is what we are going to use also!!). But there is a very important issue/puzzle.

If we use a simple Look Up Table (LUT), that stores $\mathbf{x}_i, y_i)$ and give you y_i for your ($\mathbf{x}_i$, does it minimize the above loss function? Yes; Indeed this gives zero error (perfect function) on the training set. However, a completely useless solution for the test set or for the purpose of learning. What went wrong in our problem formulation?

Really speaking, the problem we want to solve is slightly different, we would like to get a minimally complex function $f()$ that satisfy our data.

Occam's Razor: Prefer shorter hypothesis that fits the data.

- How do we define the "shortness"/complexity for the hypothesis/function $f()$.
- How do we find the shortest/shorter one?

The real problem of our interest is then

$$\min_{\mathbf{w}, f} \sum_{i \in \mathcal{D}_{Tr}} ||f(\mathbf{x}_i, \mathbf{w}) \leftrightarrow y_i|| + ||f||$$

where $||f||$ is a measure of complexity of $f()$. Without going to the details, we make certain notes here.

- The complexity of the function class $f()$ can be computed in different ways. (Say degree of a polynomial defines how complex is the polynomial. Or number of weights in a Neural network defines how complex is the neural network function.). In more formal machine learning literature, "VC" dimension is used as a complexity measure.
- Unfortunately, searching over a function class $f \in \mathcal{F}$ is not simple computationally. Searching/Trying out all neural networks and then picking the one that optimizes the above problem does not seem to be very attractive.
- In practice, we fix $f()$, and then look for $\mathbf{w}^*$. With experience you will pick a suitable function $f()$ and move forward to find the optimal $\mathbf{w}^*$. This also answers your worry about why are we trying out many solutions/hyperparameters.
- Also note that the optimization problem that we solve could be non-convex. This means that the chance of getting a good solution depends on how well we optimize and also what function class we pick.

4.20.1 Overfitting

We know that we split the data into training and testing and build the model based on the training data. There is a common serious pitfall, that is expected. The performance on training data could be very good. While performance on the test data could be very bad. This is popularly known as overfitting and should be avoided.

Overfitting Worry against overfitting is a serious concern among practitioners of machine learning. An over complex function class is more likely to overfit your training data (like LUT). This explains why do we need simple models that fit the data.

When performance of the algorithm is superior on training data and inferior on test data, we say that the algorithm is overfitting the training data.

Generalization: Though ML problems look to be very similar to the classical modelling/fitting problems with data, we always aim at doing well on the "unseen" data. Our performance is defined as the performance on the unseen data and not on the training data.

Generalization typically refers to a machine learning solution's ability to perform well on new or unseen samples rather than the training data or data that it has used/seen while trainig. It is also related to the concept of overfitting. If the model is overfitted, then it will not generalize well. We work hard to avoid overfitting.

4.21 Probabilistic View Point

Let us revisit our problem of classifying an email as spam or non-spam. It may be important to make a final classification as 0 or 1. However, in many situations what we would like to obtain is the probability of the email being spam or non-spam. This may be useful for situations like:

- a human to look closely and take the decision.
- our classification is any way under uncertainty. capture this uncertainty.
- results of this stage is used for many tasks in the subsequent stages.

Probabilistic view of the classification also allows us to incorporate the prior knowledge we have, along with the evidences we have to make optimal decisions. We will see some such examples later today.

Also a number of tools and techniques from statistics and probability theory helps us in formulating and interpretting the formulations solutions.

4.22 Terms and Definitions

We assume that a student of this course has gone through a basic course on probability theory. There are a number of terms you should recollect at this stage.

- Random Variables and Probability
- Probability Density Function
- Types of Probabilities
- Marginal Probability
- Conditional Probability
- Joint Probability
- Popular Distribution
- Normal Distribution
- Beta Distribution
- Popular Results
- Sum Rule of Probability
- Product Rule of Probability
- IID
- And many more

Do read a brief note on these associated concepts in the annexure at the end of this.

4.23 Bayes Theorem

4.23.1 Example

Let us start with an example of Bayes decision in discrete case.

You are captured by the *Sentinelese* tribe while on your excursion to the islands. You are brought to the chieftain for prosecution. You are blindfolded and the chief selects a fruit from a basket containing 85 green mangos, 5 yellow mangoes, 2 green pears and 8 yellow pears. If you guess the fruit correctly, you are set free. If not ..

- What is your guess?
- What is your chance of survival?

Simple Solution

$$P(\text{Mango}) = \frac{90}{100} = 0.9$$

$$P(\text{Pear}) = \frac{10}{100} = 0.1$$

So the safe bet is Mango. Isn't?

Evidence: Decisions are usualy not that simple. You will have more evidence to analyse the situation.

1. You get a glimpse through the blindfold and you see a slight yellow color in the chiefs hand.
2. Unfortunately you are colour-blind and you mistake green for yellow 20% of the time, but never yellow for green.

- What is your best guess?
- What will be your chance of survival now?

4.23.2 Bayes Theorem

Conditional probability : Conditional probability is the probability of observing an event, given the fact that a second event has occurred. Using formal notations, we write:
$P(fruit = mango/you\ saw\ yellow)$: read as $P(fruit = mango)$ given *you saw yellow*; or in short as: $P(mango/yellow)$.

Priori and posterior probabilities:

- Class prior probabilities $P(\omega_i)$
- In our example, this would be $P(mango)$ and $P(pear)$.
- The class-conditional probability density function $p(\mathbf{x}/\omega_i)$. The probability density function for x given the state of nature is ω_i
- In the example above the class conditional probabilities are $p(yellow/mango)$, $p(green/mango)$ etc.

Bayes Rule: Bayes rule states that the joint probability of $\mathbf{x}$ and ω_i, denoted as $p(\mathbf{x}, \omega_i)$ is given by:

$$p(\mathbf{x}, \omega_i) = p(\mathbf{x}/\omega_i).P(\omega_i) = P(\omega_i/\mathbf{x}).P(\mathbf{x})$$

We can rewrite the second equality as:

$$P(\omega_i/\mathbf{x}) = \frac{p(\mathbf{x}/\omega_i).P(\omega_i)}{P(\mathbf{x})}$$

Here the L.H.S is the posterior probability of class ω_i after observing $\mathbf{x}$. Bayes decision rule says to choose that ω_i which maximises the posterior probability. The above equation may also be written as:

$$P(\omega_i/\mathbf{x}) = \frac{p(\mathbf{x}/\omega_i).P(\omega_i)}{\Sigma_{j=1}^{c} p(\mathbf{x}/\omega_j).P(\omega_j)}$$

Bayes rules gives you a mathematical formula for combining the evidence (what you saw) with your prior knowledge (what you knew about number of fruits and their colours). The combined probability is usually called *posterior probability.*

Using Bayes rule we write:

$$P(mango/yellow) = \frac{p(yellow/mango).P(mango)}{P(yellow)}$$

Or

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{evidence}}$$

Bayes formula helps to convert the prior probability $P(\omega_i)$ to the *a posteriori* probability $P(\omega_i|x)$

Given the priors are equal, the category ω_j for which $p(x|\omega_j)$ is large is more *likely*.

If you are not colour blind:

$$P(\text{Mango/Yellow}) = \frac{\frac{5}{90} \cdot \frac{90}{100}}{\frac{5}{90} \cdot \frac{90}{100} + \frac{8}{10} \cdot \frac{10}{100}} = 0.385$$

$$P(\text{Pear/Yellow}) = \frac{\frac{8}{10} \cdot \frac{10}{100}}{\frac{5}{90} \cdot \frac{90}{100} + \frac{8}{10} \cdot \frac{10}{100}} = 0.615$$

Evidence can chnage your apriori decision!!

If you are colour blind:

$$P(\text{Mango/Yellow}) = \frac{\frac{5+0.2*85}{90} \cdot \frac{90}{100}}{\frac{5+0.2*85}{90} \cdot \frac{90}{100} + \frac{8+0.2*2}{10} \cdot \frac{10}{100}} = 0.724$$

$$P(\text{Pear/Yellow}) = \frac{\frac{8+0.2*2}{10} \cdot \frac{10}{100}}{\frac{5+0.2*85}{90} \cdot \frac{90}{100} + \frac{8+0.2*2}{10} \cdot \frac{10}{100}} = 0.276$$

4.24 Normal Distribution

We are familiar with the Normal/Gaussian distribution with mean μ and variance σ^2 from the school

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right]$$

In this case, x is the single variable. As we had seen in the problems of interest, our $\mathbf{x}$ is a vector consisting of $x_1, \ldots, x_d$. This naturally, demand the multivariate case as

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left[-\frac{1}{2}[\mathbf{x}-\mu]^T\Sigma^{-1}[\mathbf{x}-\mu]\right)$$

Indeed when $d = 1$, both these equations become the same. Naturally, our mean will be a d dimensional vector. And the covariance Σ is a $d \times d$ matrix.

$$\mu = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i$$

$$\Sigma = \frac{1}{N}[\mathbf{x}-\mu][\mathbf{x}-\mu]^T$$

- Q: What do the elements of Σ imply?
- Q: What are the properties of Σ?
- Q: How is this covariance matrix related to the correlation matrix ?
- Q: By looking at the covariance matrix, what all we can say?

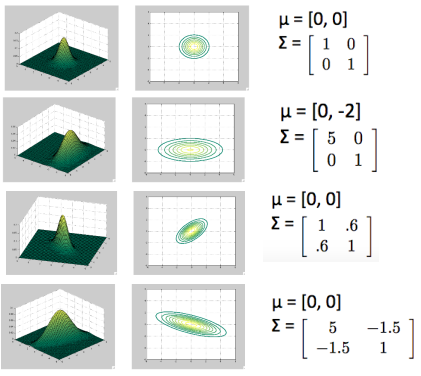


Figure 4.2: Appreciating the Covariance Matrix Structure

- Q: Why certain types of covariance matrices like $\Sigma = \sigma^2 I$ are of importance?
- We often model classes as multivariate Gaussians. Or we assume that there is an expected behaviour (measurement) for a class such as mean and there is a small deviation from the expected behavior that is modelled as Normal distribution.
- The quantity $[\mathbf{x}-\mu]^T\Sigma^{-1}[\mathbf{x}-\mu]$ is of special interest to us. This is called Mahalanobis distance.

$$\frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

$$\frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} e^{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)}$$

CSC471: Statistical Methods in AI: Lecture Note 1
A Light Introduction to Random Variables and Density Functions

Anoop M. Namboodiri
IIIT, Hyderabad, INDIA. *anoop@iiit.ac.in*

1 Introduction

Random variables are the primary mechanisms by which one deals with variability, noise and uncertainty of real-world phenomena, their observations and inferences, in statistical pattern recognition (SPR).

Consider a variable x, which represents the height of a college student in India. Let us assume that the height of a students can be anywhere between $150\ cm$ and $190\ cm$. i.e, $x \in [150, 190]$. If we randomly select a student (draw a random sample), we will get a height between 150 and 190. Hence we call x, the height of a college student in India, a *random variable*, which assumes a specific value from its range, every time we draw a sample.

Let us assume that we conduct the following experiment: We randomly select a college student from India and measure his/her height. We can ask several questions regarding the outcome of our experiment.

- How likely are we to get a student of height, say 172?
- If we repeat the experiment 500 times, how many samples will have a height greater than 180?
- Are all the height measurements equally likely?
- If not, what is the most likely value for height?
- What is the expected value of height? Is it the same as above?
- What is the expected value of height, given that the gender of the sample is, say female?

By the end of this tutorial, you should be able to answer all the above questions (and those at the end) with clear reasoning. Specifically, the last question is most interesting from the point of view of pattern classification, which asks the inverse question, "what is the most likely gender of a sample, given that the height is 165?" Before we dive deeper into the details, we introduce a few terms that will be useful through the remainder of this tutorial.

The set of all possible samples in the problem is referred to as the *population*. This could be a finite set as in the case of 'all college students in India', or an infinite set, say all possible ways in which one can write the character *'a'*. Conversely, the unit that is selected from the population in each *trial* of the experiment is referred to as a *sample*. In our example, each college student is a sample. One might consider a different experiment where each trial involves randomly selecting a set of 10 students, and the random variable x is the number of different languages that they speak. Here, each sample would be 'a set of 10 students', and the 'set of all possible subsets of size 10' from the students forms the *sample space* of x.

The process of selecting a sample is called *sampling*. The sampling process can be repeated *with replacement* or *without replacement*, depending on whether a drawn sample is put back into the population before the next sample is drawn or not. In most cases, we make the following assumptions about the samples that are drawn from a sequence of trials:

1. *Independence:* The outcome of a particular trial (the sample that is drawn) has no bearing on the outcome of the following trials. i.e, the samples are independent of each other.
2. *Identical distribution:* The probability that any particular sample is drawn is the unchanged across the trials. In other words, the probability distribution is identical for all trials.

We put the two assumptions together and claim that the samples in an experiment are *independent and identically distributed* (*i.i.d.* or *iid* for short). The above assumptions are the primary reasons why we can make any inference about a population from a relatively small set of samples drawn from the population. We often assume that the method of sampling is *(simple) random sampling*, where every sample in the population has an

equal chance of being drawn in any trial. Note that for a finite population, applying random sampling with replacement will make the resulting samples, *iid*.

In the following sections, we deal with two different types of random variables: *discrete* and *continuous*. The distinction is based on the nature of values that a random variable can take. The tools required to deal with them might also be different, which will be discussed in detail.

2 Discrete Random Variables

In our initial example, if the heights of students are measured to the nearest centimeter, the set of values that x can take are $150, 151, \ldots, 190$. x will then be a discrete random variable (DRV).

A random variable x is called a *discrete random variable*, if the set of possible values that x can take is countable. The set of values are usually finite, although not necessarily so. A discrete random variable will always take one of the n values in its range, $\chi = \{v_1, v_2, \ldots, v_n\}$ (for now, we assume n to be finite).

2.1 Probability *Mass* Function (PMF)

Now, let us consider one of our initial questions: In our random draw experiment, 'How likely are we to get a student of height 172? If our sampling is random, then the probability of drawing a sample of height 172 depends only on the number of students having height 172 in the population. Let the number of students having a height h is n_h out of the total population of N students. The probability that a randomly selected student has height h is n_h/N, as every sample has an equal probability of getting selected.

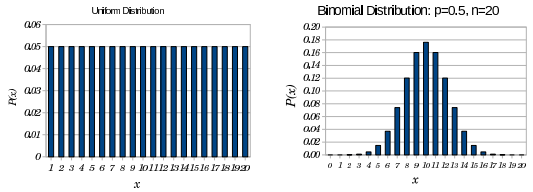
In general, the probability that a discrete random variable, x takes a value v_i (i.e., $Pr[x = v_i]$) is denoted as p_i. We denote the function that maps each value $v_i \in \chi$ to its occurrence probability, p_i, as $P(v_i)$. As we saw, the probability p_i can be computed as the fraction of the population with a value of $x = v_i$. The function $P(.)$ can be thought of as representing the distribution of the population over the values in χ, and hence is called the *Probability Distribution Function* or the *Probability Mass Function (PMF)*. To avoid confusion with similar terms, we will call $P(.)$ as the probability mass function or *PMF* in the case of discrete random variables.

As noted above, each value of p_i is a probability and the PMF should satisfy the following conditions:

$$\forall_i, P(v_i) \geq 0, and \tag{1}$$

$$\sum_{i=1}^{n} P(v_i) = \sum_{i=1}^{n} p_i = 1. \tag{2}$$

Certain parametric forms of the probability mass function are popular in practice, as they model the process of generation of the samples. Figure 1 shows two popular PMF forms, uniform and binomial.



(a) (b)

Figure 1. (a) uniform and (b) binomial distributions for PMF.

A discrete random variable is completely characterized by its PMF. Any other property of the random variable can be derived completely and precisely from its PMF. In other words, we can find the answers to all the questions posed in the introduction, if we can compute the PMF of the random variable, 'height of a college student in India'!!, well, almost all. We will now look into two of the important properties of a random variable, its expectation and variance.

2.2 Expectation and Variance: μ & σ^2

The expected value of a discrete radom variable, x is defined as:

$$\mathcal{E}[x] \equiv \mu = \sum_{x \in \chi} xP(x) = \sum_{i=1}^{n} v_i P(v_i). \tag{3}$$

What does this expectation tell us? The expected value is a *weighted average* of all possible values of x, weighted by their probabilities. In other words, μ is just the mean value of x over the entire population. Here are a couple of other ways in which we can think about μ.

- Assume that each sample in the population has unit mass, and is placed in space according to the value of x, v. The expected value, μ, will give you the centre of mass of the whole population.
- If we are asked to guess the outcome of the experiment over a large number of trials, and if we guess μ every time, we will make the least error, overall, in the MSE sense. That is why we call it the expected value.

However, if we were to guess the most likely height among all students, we will be better off guessing the mode of the distribution and not its mean . . . ofcourse!!

Now we know the best guess of the outcome of our experiment. However, can we say anything about the amount of error we will make? This is precisely what the variance tells us.

The *variance*, σ^2 of a random variable is defined as:

$$\text{Var}(x) \equiv \sigma^2 = \mathcal{E}[(x-\mu)^2] = \sum_{i=1}^{n}(v_i-\mu)^2 P(v_i). \quad (4)$$

As you can see, the variance is the mean squared error (MSE) if you guess the mean. If the *mean* tells you about the centre of mass of the population, the *variance* tells you how spread out the population is from the mean. Note that variance only gives you a measure of spread of the data and not the exact way in which it is spread. For that you need the complete PMF itself.

One can also represent the variance as:

$$\begin{aligned}
\sigma^2 &= \mathcal{E}[(x-\mu)^2] = \int_{-\infty}^{\infty}(x-\mu)^2 p(x)dx \\
&= \int (x^2 - 2x\mu + \mu^2)p(x)dx \\
&= \int x^2 p(x)dx - 2\mu \int xp(x)dx + \mu^2 \int p(x)dx \\
&= \mathcal{E}[x^2] - 2\mu\mathcal{E}[x] + \mu^2.1
\end{aligned}$$

which simplifies to:

$$\sigma^2 = \mathcal{E}[x^2] - (\mathcal{E}[x])^2. \quad (5)$$

Note: If we compute the square root of the variance, i.e, RMSE w.r.t μ, we get the *standard deviation*, σ.

3 Continuous Random Variables

In the previous section, we assumed that the height measurement of a student is an integer value, making x a discrete random variable. If we assume that the height can be measured precisely to any real number between 150 and 190, the number of values that x can take will become uncountable. Such random variables, which usually take any value within a continuous range are referred to as *continuous random variables* (CRV). Note that the number of real numbers in a range are uncountable. In each trial, the random variable, x, can take any of the infinite number of values within its range, χ. The range could also be infinite, i.e. $(-\infty, \infty)$.

In our example, even though the range is finite ($[150, 190]$), the number of possible values that x can take are uncountably infinite. This makes the definition of probabilities, tricky.

3.1 Probability *Density* Function (PDF)

Consider the continuous domain equivalent of the first question that we asked: 'If we randomly select a student, how likely are we to get a specific height, say 172.3413587391, precise up to the picometer or more?' Intuitively, we can say that it is extremely unlikely, well almost impossible, that we will chance upon a student with that exact height. i.e, $Pr[x = 172.3413587391...] = 0$. Then what about exactly $172.00...$? or any other specific real number between 150 and 190? We have to say they also have the same plight. To generalize, the probability that a continuous random variable takes any specific value in its range is 0. Does that mean no event can ever occur?!!

To get around this predicament, we reframe the question a bit as follows: 'How likely are we to select a student of height within the range $[172 - \delta, 172 + \delta]$? Now there is a non-zero probability that we might get a number within that range. Based on this, we define the distribution of samples in the range as follows:

For every continuous random variable, x, there exists a *probability density function*, $p(x)$, such that:

$$\forall_x, p(x) \geq 0, \quad and \quad (6)$$

$$Pr[x \in (a,b)] = \int_a^b p(x)dx. \quad (7)$$

From the second condition, we can also infer that:

$$\int_{-\infty}^{\infty} p(x)dx = 1 \quad (8)$$

$p(x_t)$ gives the limiting value for density of probability in a small window around the point x_t. Note that the value of $p(x_t)$ is not a probability. We always use lower case p for densities, and upper case P for functions that give probabilities. The probability density function (PDF), plays the same role for CRVs as PMF for DRVs. Note that in theory the PDF need not be a parametric function, although in practice it always is.

3.2 Expectation and Variance: μ & σ^2

We can extend the definitions of expected value and variance of a RV from the discrete to continuous domain as the following integrals:

$$\begin{aligned}
\mathcal{E}[x] &\equiv \mu &&= \int_{-\infty}^{\infty} xp(x)dx \\
\text{Var}(x) &\equiv \sigma^2 &&= \int_{-\infty}^{\infty}(x-\mu)^2 p(x)dx.
\end{aligned}$$

These measures also have similar meanings or interpretations as we found for the discrete RVs. To make the ideas clear, we will consider two examples of PDFs:

3.2.1 Uniform Density

The uniform density function is characterized by the range within which it is defined as is given by:

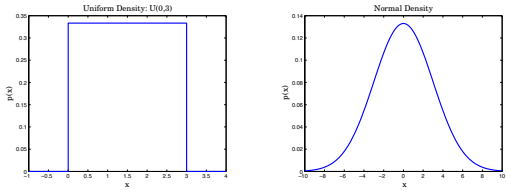
$$U(a,b) = \begin{cases} \frac{1}{(b-a)} & \text{if } a \leq x \leq b \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

$$\begin{aligned} \mu &= \int_a^b x \left(\frac{1}{b-a}\right) dx = \frac{1}{b-a}\int_a^b x dx \\ &= \frac{1}{b-a}\left[x^2/2\right]_a^b = (b+a)/2, \end{aligned}$$

which is what we expect of the mean of a uniform distribution between a and b. Similarly, the variance can shown to be:

$$\sigma^2 = \frac{(b-a)^2}{12} \tag{10}$$

Figure 2(a) shows the plot of a uniform density function in the range $[0, 3]$.



(a) (b)

Figure 2. (a) uniform and (b) norma densities for PDF.

3.2.2 Normal Density

The Normal or Gaussian density is one of the most popular density functions in practice, as it is a good approximation of many real world random processes. The normal density function, $N()$ has two parameters, μ, and σ, and is given by:

$$N(\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{11}$$

Figure 2(b) shows a normal density plot: $N(0, 3)$, within the range $[-10, 10]$. Note that the support of a normal density if infinite. The expectation and variance of the normal density function are in fact, μ and σ^2 themselves.

In addition to what we discussed, there are a large number of probability distributions for both discrete and continuous RVs that are used in specific scenarios [1].

4 CDF: Cumulative Distribution Function

The cumulative distribution function or CDF is derived from the PDF by the integral of the density up to a point. It is defined as:

$$C(t) = \int_{-\infty}^{t} p(x)dx. \tag{12}$$

Not that the CDF gives the total probability that a continuous random variable takes a value less than a specific value, t. The CDF is can be expressed in a parametric form in certain cases, such as the uniform density:

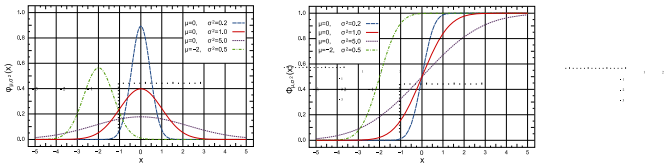
$$C(t) = \begin{cases} 0 & \text{if } t < a \\ \frac{(t-a)}{(b-a)} & \text{if } a \leq t \leq b \\ 1 & \text{if } t > b \end{cases} \tag{13}$$

Note that a PDF of a RV completely specifies its CDF and vice-versa. However, it is possible that one of them has a compact parametric representation, while the other does not. For example, the CDF of the normal distribution (equation 11) is given by:

$$cdf(x) = \frac{1}{2}\left(1 + erf\left(\frac{x-\mu}{\sigma\sqrt{2}}\right)\right), \tag{14}$$

where $erf()$ is the error function defined by:

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_{-\infty}^{x} e^{-t^2} dt. \tag{15}$$



(a) (b)

Figure 3. (a) normal densities with different parameters and (b) the their CDFs [1].

There is no closed form representation to the erf, and it is often approximated by its Taylor series expansion. Figure 3 shows the normal density function with four different parameters, and the corresponding cumulative distribution functions.

4.1 Generating Random Numbers

One of the very useful applications of CDFs is that one can generate random numbers that follow any given distribution, provided we can compute/estimate the CDF of the distribution.

Consider a random variable, x that is distributed according to a PDF, $p(x)$. Also consider another random variable, $y = C(x)$, where $C(x)$ is the CDF corresponding to $p(x)$.

Now, consider a small window of x around the point t, $[t - \delta t, t + \delta t]$ (see Figure 4). The value of y corresponding to t will be $r = C(t)$. Moreover, the value of y corresponding to $x = t+\delta t$ will be $r+\delta t.p(t)$, assuming that δt is small, giving $p(t + \delta t) \approx p(t)$. Similarly, $C(t - \delta t) = r - \delta t.p(t)$.

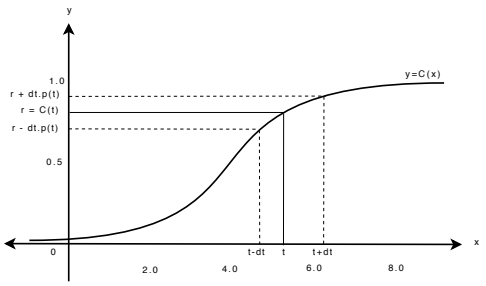


Figure 4. Mapping of a random variable using the CDF.

In other words, all samples of x within a window of size $2\delta t$ around t will map to a window of size $2\delta t.p(t)$ around $C(t)$ for y. The resulting density of y will be hence $1/p(t)$ times the density of x, which is unity. i.e, y is of uniform density in the range $[0, 1]$.

We just argued that given a random variable x of any density, the corresponding random variable, $y = C(t)$ will be $U[0, 1]$. We can invert this statement and say that given a random variable y that follows the pdf $U[0, 1]$, the random variable $x = C^{-1}(y)$ will follow a PDF with corresponding CDF as $C()$. In other words given a set of random numbers y_i with uniform density $U(0, 1)$, we can map it to a set of random variables x_i with any desired PDF using the inverse CDF function !!!

5 Problems

1. Give an example each of probability mass functions with finite and infinite ranges. Show that the conditions on PMF are satisfied by your example.
2. Show with complete steps that the variance of uniform density is given by equation 10. (*Hint: use the expression for variance in equation 5.*)
3. Show examples of two density functions (draw the function plots) that have the same mean and variance, but clearly different distributions. Plot both functions in the same graph with different colours.
4. Show that the alternate expression for variance given in equation 5 holds for discrete random variables as well.
5. Prove that the mean and variance of a normal density, $N(\mu, \sigma^2)$ are indeed its parameters, μ and σ^2.
6. Using the inverse of CDFs, map a set of $10,000$ random numbers from $U[0, 1]$ to follow the following pdfs:
   (a) Normal density with $\mu = 0$, $\sigma = 3.0$.
   (b) Rayleigh density with $\sigma = 1.0$.
   (c) Exponential density with $\lambda = 1.5$.

   Once the numbers are generated, plot the normalized histograms (the values in the bins should add up to 1) of the new random numbers with appropriate bin sizes in each case; along with their pdfs. What do you infer from the plots? *Note: see rand() function in C for $U[0, INT\_MAX]$.*
7. Write a function to generate a random number as follows: Every time the function is called, it generates 500 new random numbers from $U[0, 1]$ and outputs their sum.

   Generate $50,000$ random numbers by repeatedly calling the above function, and plot their normalized histogram (with bin-size = 1). What do you find about the shape of the resulting histogram?

References

[1] *Probability distribution*, Wikipedia, 2008, http://en.wikipedia.org/wiki/Probability_distributions.

[2] R. Duda, P. Hart, and D. Stork, *Pattern classifcation and scene analysis*, 2^{nd} ed., John Wiley and Sons, New York, 2001.

[3] John A. Rice, *Mathematical statistics and data analysis*, 2nd ed., Duxbury Press, 1995.

CSE 475: Statistical Methods in AI **Monsoon 2019**

SMAI-M-2019 6: Mathematical Foundations of ML - VI

Lecturer: C. V. Jawahar *Date: 19 Aug 2019*

6.31 Problem Space - VI

Recall that $(x_i, y_i), i = 1, ...n$ are independent pairs with the same distribution as (X, Y), and that $f()$ fit on this training set. We will look at the expected test error, conditional on $X = x$ for some arbitrary input x

6.31.1 Bias Variance Tradeoff

The expected test error in, also called the prediction error, for any machine learning algorithm can be broken down into three parts:

1. Bias Error
2. Variance Error
3. Irreducible Error

The third component, we may not be able to help. But we can adjust our problem formulation (say hyper parameters or model complexity) that can give us a solution that has lower error. From a conceptual level, we will see what are these errors at this stage. We will see how they practically appear in subsequent lectures.

We can never predict y_i from x_i with zero prediction error. Note that what we are interested is in predicting for the future. Even if have a magical function that capture the ideal relation underlying X and Y, i.e., the true function, $y_i = f(x_i)$ or $f(X) = E(Y|X)$, we would still incur some error, due to noise in practical situations. We call this as irreducible error. This error can not be reduced regardless of the algorithm we use. We are helpless. It is the error introduced from the modeling/formulation of the problem, and may be the result of factors like unknown variables etc.

What happens if our fitted function $f()$ belongs to a model class that is far from the true function $f()$? E.g., we choose to fit a linear model in a setting where the true relationship is far from linear, say quadratic? We will often refer to this as a wrong (or overcomplex) model. As a result of this design choice, we encounter an error. We call this as an estimation bias. and error due to bias.

What happens if our fitted (random) function $f()$ is itself quite variable? In other words, over different subsets/sampling of the training set, we end up constructing a substantially different functions $f()$? This is also bad. This is another source of error. We will call this estimation variance.

Bias Error

- Low Bias: Suggests less assumptions about the form of the target function.
- High-Bias: Suggests more assumptions about the form of the target function.

Variance Error

- Low Variance: Suggests small changes to the estimate of the target function with changes to the training dataset.
- High Variance: Suggests large changes to the estimate of the target function with changes to the training dataset.

There is a trade off here, but it need it is really never be one-to-one; i.e., in some cases, it can be worth sacrificing a little bit of bias to gain large decrease in variance, and in other cases,vice versa Typical trend: underfitting means high bias and low variance, overfitting means low bias but high variance.

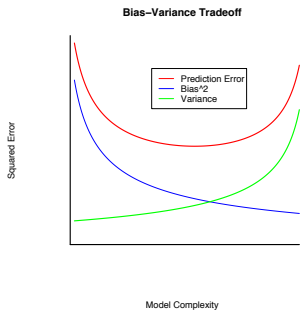


Figure 6.3: The tradeoff of Bias and Variance

6.32 Unifying View Points

Let us now look at the three different view points that we have for the classification.

- Generative view
- Discriminative view
- Distance based view

We start with a simple problem of binary classification with one varible (univariate gaussian with equal variance)

6.32.0.1 Generative/Probabilistic view

We look at the solution as decide class 1 if $P(\omega_1|x) > P(\omega_2|x)$ else class 2.

If we plot the class denisties as Gaussians, with mean as μ_1 and μ_2, we see that this classification makes all samples with $x < \theta$ as class 1 and others class 2. And

$$\theta = \frac{\mu_1 + \mu_2}{2}$$

This is the Bayesian optimal classifier.

6.32.0.2 Discriminative View

We had seen classifiers that say $\mathbf{w}^T\mathbf{x} > \theta$, then class 1 else class 2. Our classifier of $[1][x] > \theta$ is exactly the same.

We see that the simple linear classifier like $\mathbf{w}^T\mathbf{x} > \theta$ is identical to the Bayesian optimal ones under certain settings.

6.32.0.3 Distance bassed view

Assume we look at the classification problem as "classify x to class 1 if distance between x and μ_1 is smaller than that of distance between x and μ_2. This also reduces to the same classifier we discussed above.

In short, all three different looking paradigms yield the same results (under certain assumption).

6.32.1 Multivariate Gaussians

The arguments for the univariate will remain same even if move to Multivariate setting (say with identical covariance).

Q: Do you see a role for Mahalanobis distance?

6.33 Parameter Estimation

Consider the problem of estimating parameters given a data set. Why is this important? Note that we often have only discrete samples and not a distribution to start with. We can make assumption of the type of distribution. But the parameters could be unknown.

Problem Setting There is a training set $\mathcal{D} = \{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_m}, y_m)\}$ of examples sampled i.i.d. according to $p(\mathbf{x}, y)$.

What is IID? This is a popular and valid assumption to make in most situations.

- **Independent:** Each example is sampled independently from the others.
- **Identically Distributed:** All examples are sampled from the same distribution.

Consider the situation when all the samples come from c different samples (like a multi class classification problem). The training set can be divided into $\mathcal{D}_1 \ldots, \mathcal{D}_c$ subsets, one correspond to each of these classes ($\mathcal{D}_i = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ contains i.i.d examples for target class y_i)

For any new example $\mathbf{x}$ (not in training set), we compute the posterior probability of the class given the example and the full training set $\mathcal{D}$ as

$$P(y_i|\mathbf{x}, \mathcal{D}) = \frac{p(\mathbf{x}|y_i, \mathcal{D})p(y_i|\mathcal{D})}{p(\mathbf{x}|\mathcal{D})}$$

Note that this is very similar to the Bayesian decision theory (compute posterior probability of class given example) except that parameters of distributions are unknown, and a training set $\mathcal{D}$ is provided instead.

We assume $\mathbf{x}$ is independent of $\mathcal{D}_j (j \neq i)$ given y_i and $\mathcal{D}_i$ with out additional knowledge, $p(y_i|\mathcal{D})$ can be computed as the fraction of examples with that class in the dataset the normalizing factor $p(\mathbf{x}|\mathcal{D})$ can be computed marginalizing $p(\mathbf{x}|y_i, \mathcal{D}_i)p(y_i|\mathcal{D})$ over possible classes.

6.33.1 Estimating θ

We must estimate class-dependent parameters θ_i for $p(\mathbf{x}|y_i, \mathcal{D}_i)$. Assumes parameters θ_i have fixed but unknown values, Values are computed as those maximizing the probability of the observed examples $\mathcal{D}_i$ (the training set for the class). Obtained values are used to compute probability for new examples

$$p(\mathbf{x}|y_i, \mathcal{D}_i) = p(x|\theta_i)$$

Bayesian Parameter Estimation: Assumes parameters θ_i are random variables with some known prior distribution. Observing examples turns prior distribution over parameters into aposteri distribution. Predictions for new examples are obtained integrating over all possible values for the parameters

$$p(\mathbf{x}|y_i, \mathcal{D}_i) = \int_{\theta_i} p(\mathbf{x}, \theta_i|y_i, \mathcal{D}_i)d\theta_i$$

6.33.2 Maximum a-posteriori Estimation

$$\theta_i^* = \arg\max_{\theta_i} p(\theta_i|\mathcal{D}_i, y_i) = \arg\max_{\theta_i} p(\mathcal{D}_i, y_i|\theta_i)p(\theta_i)$$

Assumes a prior distribution for the parameters $p(\theta_i)$ is available

6.33.3 Maximum Liklihood Estimation

$$\theta_i^* = \arg\max_{\theta_i} p(\mathcal{D}_i, y_i|\theta_i)$$

maximizes the likelihood of the parameters with respect to the training samples. Here, no assumption about prior distributions for parameters is made.

Each class y_i is treated independently: replace $y_i, \mathcal{D}_i \rightarrow \mathcal{D}$ for simplicity.

- A training data $\mathcal{D} = \mathbf{x}_1, \ldots, \mathbf{x}_N$ of i.i.d. examples for the target class is available.
- We assume the parameter vector θ has a fixed but unknown value.
- We estimate such value maximizing its likelihood with respect to the training data

$$\theta^* = \arg\max_{\theta} p(\mathcal{D}|\theta) = \arg\max_{\theta_n} \prod_{j=1}^{N} p(x_j|\theta)$$

- The joint probability over $\mathcal{D}$ decomposes into a product as examples are i.i.d (thus independent of each other given the distribution)

Maximum log-likelihood It is usually simpler to maximize the logarithm of the likelihood (monotonic)

$$\theta^* = \arg\max_{\theta} \ln p(\mathcal{D}|\theta) = \arg\max_{\theta} \sum_{j=1}^{n} \ln p(x_j|\theta)$$

Necessary conditions for the maximum can be obtained zeroing the gradient with respect to to θ

$$\nabla_\theta = 0$$

Note zeroing the gradient can be local or global maxima depending on the form of the distribution

6.34 Example: Gaussians

Now we take an example where we want to estimate the parameters of a distribution, given a set of samples. We know the univariate and multivariate Gausiians as: Univariate

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right]$$

and multivariate case

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left[-\frac{1}{2}[x-\mu]^T\Sigma^{-1}[x-\mu]\right]$$

Note that the final expression for the parameter θ (in this case, mean and covariance) may be familiar to you. But the point to note is that, this leaves a Bayesian perspective of how parameters are estimated.

6.34.1 Unknown μ

Let us take a simple case, where covariance Σ is known and only mean μ is unknown.

The log-likelihood is

$$\sum_{j=1}^{n} \ln p(\mathbf{x}_j|\theta) - \sum_{j=1}^{n} \frac{-1}{2}[\mathbf{x}_j-\mu]^T\Sigma^{-1}[\mathbf{x}-\mu]$$

The gradient with respect to the mean is

$$\nabla \sum_{j=1}^{n} \ln p(\mathbf{x}_j|\theta) = \sum_{j=1}^{n} \Sigma^{-1}[\mathbf{x}_j-\mu]$$

Setting the gradient to zero gives

$$\sum_{j=1}^{n} \Sigma^{-1}[\mathbf{x}_j-\mu] = 0 \implies \mu^* = \frac{1}{n}\sum_{j=1}^{n}\mathbf{x}_j$$

6.34.2 Unknown μ and σ^2

The log-likelihood is

$$\mathcal{L} = \sum_{j=1}^{n} -\frac{1}{2\sigma^2}(x_j-\mu)^2 - \frac{1}{2}\ln 2\pi\sigma^2$$

The gradient wrt to μ and σ^2 of L respectively are

$$\sum_{j=1}^{n} \frac{1}{\sigma^2}(x_j-\mu)$$

and

$$\sum_{j=1}^{n} \left(-\frac{1}{2\sigma^2} + \frac{(x_j-\mu)^2}{2\sigma^4}\right)$$

Setting gradients to zero lead to

$$\mu^* = \frac{1}{n}\sum_{j=1}^{n}\mathbf{x}_j$$

and

$$\sum_{j=1}^{n} \frac{1}{2\sigma^2} = \sum_{j=1}^{n} \frac{(x_j-\mu)^2}{2\sigma^4}$$

$$\sum_{j=1}^{n} \sigma^2 = \sum_{j=1}^{n} (x_j-\mu)^2$$

$$\sigma^2 = \frac{1}{n}\sum_{j=1}^{n}(x_j-\mu)^2$$

6.34.3 Unknown μ and Σ

The log-likelihood is:

$$\sum_{j=1}^{n} -\frac{1}{2}[\mathbf{x}_j-\mu]^T\Sigma^{-1}[\mathbf{x}_j-\mu] - \frac{1}{2}\ln(2\pi)^d|\Sigma|$$

The maximum-likelihood estimates are:

$$\mu^* = \frac{1}{n}\sum_{j=1}^{n}\mathbf{x}_j$$

and

$$\Sigma = \frac{1}{n}\sum_{j=1}^{n}[\mathbf{x}_j-\mu^*][\mathbf{x}_j-\mu^*]$$

Maximum likelihood estimates for Gaussian parameters are simply their empirical estimates over the samples:

- Gaussian mean is the sample mean
- Gaussian covariance matrix is the mean of the sample covariances

Q: Verify the above equations/derivations

CSE 475: Statistical Methods in AI **Monsoon 2019**

SMAI-M-2019 7: Linear Models(I): Regression

Lecturer: C. V. Jawahar *Date: 26 Aug 2019*

7.35 Background

We had seen how a typical ML problems gets formulated. We are looking for a data driven model that can predict y_i given $\mathbf{x}_i$. i.e.,

$$y_i = f(\mathbf{x}, \mathbf{w})$$

Our solution is parameterized by $\mathbf{w}$. During training, an appropriate optimization problem was solved to obtain $\mathbf{w}$ from $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}, \quad i = 1, \ldots, N$ We also appreciated the basic mathematical notions associated with these problems in the last set of lectures. To recollect, we talked about problems and the solutions obtained by solving an optimization problem over the data during the training. What were our problems?

Classification In classification problems y_i is an integer. What value we assign is of not much significance. For example, some people use 0 and 1, while some where else we use -1 and $+1$ as the class IDs. Multi class classification where the number of classes is more than 2 is a popular case. Though multiclass classification is very popular and important in practice, many discussions in the linear classifiers assume that the number of classes is only 2. That makes the life simple. (Don't worry. We can extend these ideas to multiclass setting without much effort. We will also see some later on.)

Regression In the case of regression, y_i is a real quantity. It could be a real vector or a simple real number. Let us assume it as a real number at this stage.

Others There are many other situations where the space of y has structure. For example, y is a graph or a string. Such problems are beyond our interest at this stage. Popularly these are also called structured prediction problems.

7.36 Modelling and Solving

Now we will systematically look into two very important aspect of the machine learning:

- How do we model the problem and relationship between the variables? (or how do we choose the function class of interest)
- How do we formulate and solve the associated optimization problem for training? What are the associated theoretical issues in optimization? and what are the associated practical issues in implementation?

We will see some of these in the next few lectures.

Linear Vs Non-Linear Models Two major classes of models that we use are:

- Linear Models: Simple models; yet effective in many problems; linear functions.
- Non-Linear Models (eg. Neural Networks): Very effective for complex problems; Hard to interpret

Convex and Non-Convex Optimization Some of the optimization problems lead to globally optimal solutions and performance guarantees. Based on the nature of the objective function and constraints, problems are broadly classified into:

- Convex optimization (eg. MSE in linear regression, SVMs)
- Non-Convext optimization (eg. Training Deep Neural Networks)

7.37 Linear Models in ML

Let us first start with a specific, (simple), and yet effective class of models/functions.

$$y = f(x) = \mathbf{w}^T\mathbf{x} + w_0 \tag{7.7}$$

If $d = 2$ then the model is something like

$$y = w_2x_2 + w_1x_1 + w_0$$

The additional term w_0 is required to make sure that it covers all the possible "lines" including the ones that does not pass through the origin.

This is not limited to 2D samples. Lines naturally gets extended to planes and hyperplanes. For example

$$y = w_3x_3 + w_2x_2 + w_1x_1 + w_0$$
$$y = w_4x_4 + w_3x_3 + w_2x_2 + w_1x_1 + w_0$$

The notation (and many math that come later) could be simpler if we do not have w_0. We do that by augmenting the vector $\mathbf{x}$ with an additional quantities. i.e., the new $\mathbf{x}$ is the old $\mathbf{x}$ with an additional 1 concatenated at the end. Similarly the $\mathbf{w}$ is augmented with w_0 and the equation 7.7 gets simplified as

$$y = f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T\mathbf{x} \tag{7.8}$$

Note that we did not introduce a different notation with and without augmentation. This is to make the notations simpler. (Text books may be using different notations for these two). Hope you appreciate the convenience of augmenting with 1.

What more we can do with augmentation (or explicit feature maps)? We can infact create a new vector from the old ones. This also generalizes the trick of augmentation.

Consider that our original vector is $[x_1, x_2]^T$. We now know how to create a new vector $[x_1, x_2, 1]$. Why not

$$[x_1, x_2]^T \rightarrow [x_1^2, x_2^2, x_1x_2, x_1, x_2, 1]^T$$
$$\phi(\mathbf{x}) = [x_1^2, x_2^2, x_1x_2, x_1, x_2, 1]^T$$

Such a modification will allow us to learn a new model $\mathbf{w}^T\mathbf{x}$ as

$$w_5x_1^2 + w_4x_2^2 + w_3x_1x_2 + w_2x_1 + w_1x_2 + w_0 \tag{7.9}$$

This is really a quadratic function. Our linear algorithm (that we see soon) is able to learn nonlinear models too, provided $\mathbf{x}$ was augmented and modified to suite this function. The objective of introducing this feature mapping trick ($\phi()$) at this stage is to convince you that the algorithms that we will discuss are very powerful and useful. Linearity does not constrain us too much. We will see this feature map again when we discuss kernels in the context of SVMs.

7.38 Regression and MSE

7.38.1 Line Fitting

We want to start with a simple problem. We are given a number of samples $x_i, y_i)$ $i = 1, \ldots, N$ and our objective is to find a line that fits this data. i.e.,

$$y_i = w_1x_i + w_0$$

Note our problem is to find two coefficients (parameters) w_1 and w_0 that can define the line. Indeed when we fit a line, all points need not be on the line. There could be errors. This leads to

$$y_i = w_1x_i + w_0 + \epsilon_i$$

What should be our goal? We want ϵ_i to be small for all the samples or

$$Minimize \frac{a}{N}\sum_{i=1}^{N}\epsilon_i^2$$

Let us now see how do we find the best $\mathbf{w}$ for our line fitting problem in 1D. i.e., (x_i, y_i). As discussed above, by augmenting, we got the problem as $(\mathbf{x_i}, y_i)$. We want a model $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$. Or we want to minimize the error $\epsilon_i = y_i - \mathbf{w}^T\mathbf{x}$. Let us write the equations as vectors/matrices.

Though we started with the problem of fitting a line (linear model) in 2D plane, the final problem we have formulated is applicable for any data in d-dimensions. Lines become hyperplane. y_i still remain scalar.

$$Minimize \frac{1}{N}\sum_{i=1}^{N}\epsilon_i^2 = Minimize \frac{1}{N}\sum_{i=1}^{N}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

7.38.2 Mean Squared Error (MSE) Loss

There are N samples with us. We first state our problem a compact matrix form.

$$\begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_N \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} - \begin{bmatrix} \mathbf{x_1}^T \\ \mathbf{x_2}^T \\ \vdots \\ \mathbf{x_N}^T \end{bmatrix} \mathbf{w} \tag{7.10}$$

The equation is more compactly written as

$$\mathbf{E} = \mathbf{Y} - \mathbf{X}\mathbf{w}$$

The loss (sum of squared error over all the samples) or mean sum of sqaured error (MSE) is:

$$J = \frac{1}{N}\mathbf{E}^T\mathbf{E}.$$

Why are we dividing by N?

Our objective is to minimize the above loss. What do we have control to modify? $\mathbf{w}$. We minimize over $\mathbf{w}$. i.e, $J()$ is a function of $\mathbf{w}$ and the minima can be obtained by optimizing

$$J(\mathbf{w}) = \frac{1}{N}[\mathbf{Y} - \mathbf{X}\mathbf{w}]^T[\mathbf{Y} - \mathbf{X}\mathbf{w}]$$

where $\mathbf{Y}$ is a $N \times 1$ vector. $\mathbf{X}$ is a $N \times d$ matrix and $\mathbf{w}$ is a $d \times 1$ vector.

$$J(\mathbf{w}) = \frac{1}{N}(\mathbf{Y}^T\mathbf{Y} + \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w} - 2\mathbf{w}^T(\mathbf{X}^T\mathbf{Y}))$$

7.38.3 Closed form solution

Our problem is to find the "best" $\mathbf{w}$. Note that the only variable in the loss function is $\mathbf{w}$. We can differentiate the loss function with respect to $\mathbf{w}$ and equate to zero to get the minima. This leads to

$$2\mathbf{X}^T\mathbf{X}\mathbf{w} - 2\mathbf{X}^T\mathbf{Y} = 0$$

or

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$$

Since we started by finding a $\mathbf{w}$ that suits $\mathbf{Y} = \mathbf{X}\mathbf{w}$. We can also look $(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T$ as the pseudo inverse of $\mathbf{X}$.

- Q:Why do we have to have a pseudo inverse? Why not a regular inverse?
- Q:Under what situations the inverse in the above equation can not be computed? When can this happen for the above MSE problem.
- Q;If all the samples were on the line itself. i.e., all the errors e_is were zero. What do we know about the matrix $(\mathbf{X}^T\mathbf{X})$?
- Q: If $N = 1$, is the problem (easily) solvable? what could happen to ths solution?

7.38.4 Discussions

The above formulation is very effective. However, here are some points woth noting.

- This assume all the samples are available before we start. (not when samples come over time in an online manner).
- The inverse of a $d \times d$ matrix is not very attractive. Specially when d is large.
- As a line fitting in 2D, this is not very intuitive, some time we want the orthogonal distance to the line to be minimized.

7.39 Probabilistic View Point

We saw how a closed form solution to the regression with MSE loss can be obtained. However, is there a rationale for the MSE loss? Here, we argue that the MLE estimate of the parameters, under certain assumptions, lead to MSE loss/objective.

We know that the problem is modelled as

$$y_i = \mathbf{w}^T\mathbf{x} + \epsilon_i$$

We assume that the ϵ_i are distributed IID (Independently and Identically Distributed) according to a Gaussian/Normal distribution with zero mean and variance σ^2

$$p(\epsilon_i) = \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{\epsilon_i^2}{2\sigma^2}\right)$$

This leads to

$$p(y_i|\mathbf{x}_i, \mathbf{w}) = \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{(y_i - \mathbf{w}^T\mathbf{x})^2}{2\sigma^2}\right)$$

This represents the distribution of y_i given $\mathbf{x_i}$ and parameterized by $\mathbf{w}$. We model this as $\mathcal{N}(\mathbf{w}^T\mathbf{x}, \sigma^2)$

Our problem is now to find the MLE estimate of $\mathbf{w}$ from the data. Given the data $(\mathbf{X}, Y)$, we can do a maximum likelihood estimate of $\mathbf{w}$.

$$L(\mathbf{w}) = L(\mathbf{w}, \mathbf{X}, \mathbf{Y}) = \prod_{i=1}^{N} p(y_i|\mathbf{x}_i, \mathbf{w})$$

$$= \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{(y_i - \mathbf{w}^T\mathbf{x})^2}{2\sigma^2}\right)$$

Instead of maximizing $L(\mathbf{w})$, we maximize the log-likelihood.

$$l(\mathbf{w}) = \log\ L(\mathbf{w}) = \log \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{(y_i - \mathbf{w}^T\mathbf{x})^2}{2\sigma^2}\right)$$

$$= \sum_{i=1}^{N} \log \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{(y_i - \mathbf{w}^T\mathbf{x})^2}{2\sigma^2}\right)$$

$$= K\frac{1}{2}\sum_{i=1}^{N}(y_i - \mathbf{w}^T\mathbf{x})^2$$

Where K is a constant.

This imply now that the MSE objective we had is only a scaled version of the MLE estimate objective we have here. Both view points come closer again.

$$\Rightarrow \min_{\mathbf{w}} J(\mathbf{w}) = \min_{\mathbf{w}} \sum_{i=1}^{N}(y_i - \mathbf{w}^T\mathbf{x}_i)^2$$

In other words, under the appropriate probabilistic assumptions of the data, least-squares linear regression (MSE) is exactly the same as the maximum likelihood estimate of $\mathbf{w}$.

Another interesting point to note at this stage is that our final choice of $\mathbf{w}$ did not depend on what was σ^2.

7.40 Lasso and Ridge Regression

Let us revisit our regression model

$$y_i = \mathbf{w}^T \mathbf{x}_i + \epsilon_i$$

and our objective

$$J = \frac{1}{2} \sum_{i=1}^{N} (y_i - \mathbf{w}^T \mathbf{x})^2.$$

The solution to this was arrived as

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y}$$

Bias and variance If we had sampled many different sub-sets of samples and did linear regression, say for M times, what we should have obtained? Indeed we should have obtained M different estimates of the regression coefficients $\mathbf{w}_i$ and errors J_i for $i = 1, \ldots, M$. Note that we often are limited by a small number of samples to work with.

- Variance is the amount by which estimate of the solution/function will change if a different training subset was employed for training.
- Bias is known as the assumptions made by a model or designer to make the target function simpler to learn. Low bias suggests less assumptions about the form of the target function. High-bias suggests more assumptions about the form of the target function.

We are interested in errors from bias and variance. The bias is an error from erroneous assumptions in the learning algorithm. It simply means how far away is our estimated values from actual values. The variance is an error from sensitivity to small fluctuations in the training set. It is a measure of spread or variations in our predictions. High variance can cause an algorithm to model the random noise in the training data, rather than the intended outputs (overfitting).

To build models that can generalize well and reduce errors on the test data (unseen data), we need to improve the simple regression we learned.

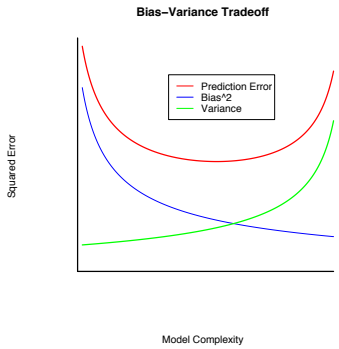


Figure 7.2: Bias-Variance View

Ridge Regression In ridge regression, we improve the objective as

$$J = \sum_{i=1}^{N} (y_i - \mathbf{w}^T \mathbf{x})^2 + \lambda \sum_{j=1}^{d} w_j^2$$

$$= \sum_{i=1}^{N} (y_i - \mathbf{w}^T \mathbf{x})^2 + \lambda ||\mathbf{w}||_2^2.$$

What happens when λ is very low (say zero) and very high (say infinity)? When λ is zero, this reduces to the normal MSE based regression and when λ is infinity, $\mathbf{w}$ reduces to zero. Both these extreme cases need to be the best. We need to find the best λ for our data/problem. (See figure). This is often done by cross validation.

Let us come back to ridge regression objective. It can be done as:

$$J(\mathbf{w}) = \frac{1}{N} (\mathbf{Y}^T \mathbf{Y} + \mathbf{w}^T \mathbf{X}^T \mathbf{X} \mathbf{w} - 2\mathbf{w}^T (\mathbf{X}^\mathbf{T} \mathbf{Y} + \lambda \mathbf{w}^T \mathbf{w}))$$

Optimization of the above objective function leads to:

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X} + \lambda I)^{-1} \mathbf{X}^T \mathbf{Y}$$

- The bias increases as λ (amount of shrinkage) increase.
- The variance decreases as λ increases

(See figure)

Lasso Regression Lasso is another variant of the regression where the objective has $L1$ norm of the $\mathbf{w}$ instead of $L2$.

$$J = \sum_{i=1}^{N} (y_i - \mathbf{w}^T \mathbf{x})^2 + \lambda \sum_{i=1}^{d} |w_j|$$

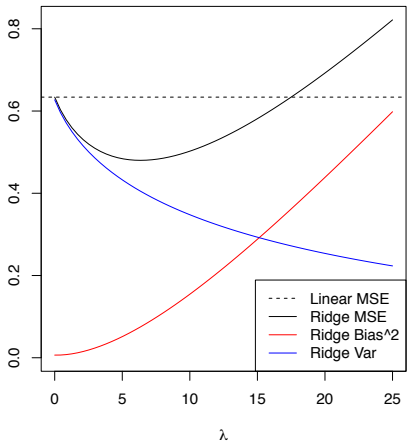


Figure 7.3: Bias-Variance View of Ridge Regression

$$= \sum_{i=1}^{N} (y_i - \mathbf{w}^T \mathbf{x})^2 + \lambda ||\mathbf{w}||_1.$$

In Lasso, we not only penalize the high value of some of the coefficients, optimization also leads to zero for irrelevant variables.

The change in the norm of the penalty may seem like only a minor difference, however the behavior of the $L1$-norm is significantly different than that of the $l2$-norm.

7.41 Challenges with Data Size

7.41.1 Incremental Setting and Large Data Setting

In the last sections, we saw the problem of regression with Mean Square Error (MSE) Loss/Objective getting solved directly with a closed form solution as:

$$\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{Y}$$

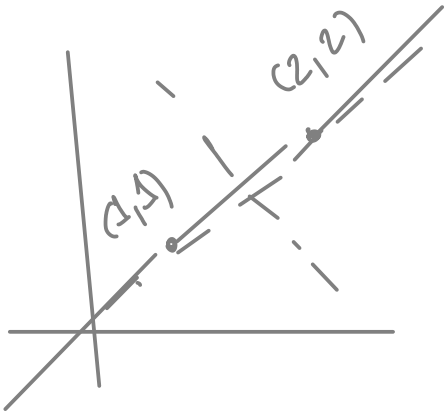
what all influence the computational complexity of this solution?

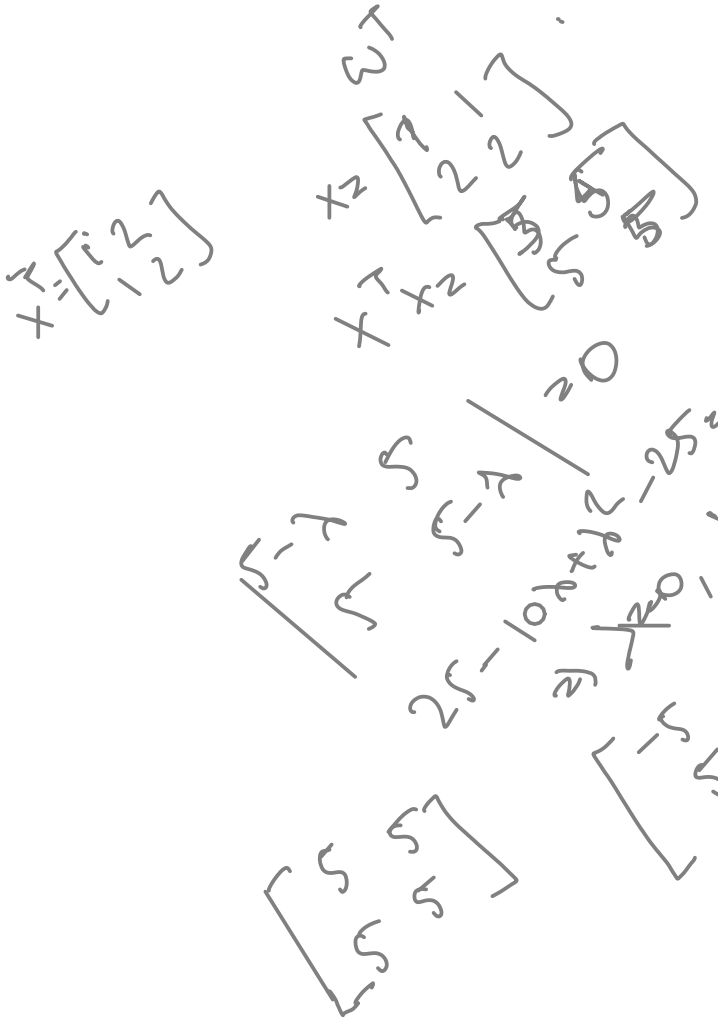
- When N is very large? Do we have to use all the data?
  - Stochastic Versions
  - Work on subsets that on full data at a time
- When N is large and computational capabilities are limited
  - Iterative solution scheme → "Gradient Discent"
- When all the data can not be stored/available
  - Online variants (see home works)

These issues, lead to the settings like:

Large Data Sets Where does the number of samples N come into the picture?

Incremental Setting Another situation of importance is when we get newer samples on a regular basis.





CSE 475: Statistical Methods in AI **Monsoon 2019**

SMAI-M-2019 8: Linear Models(II): PCA

Lecturer: C. V. Jawahar *Date: Aug 29, 2019*

8.43 A Related Problem

In the last lecture we looked at the line fitting when we were predicting y_i from $\mathbf{x_i}$. We minimized an error in y_i given $\mathbf{x}_i$. Something like:

$$\sum_{i=1}^{N}(y_i - f(\mathbf{w}, \mathbf{x}_i))^2$$

If you visualize this in a 2D plane (when $\mathbf{x}$ was 1D) the error is parallel to the y axis. Or axis parallel. In general, the error is always defined with respect to y.

Consider a set of points in 2D $i\{(x^1, x^2)_i\}$. A very related problem is how do we fit a line that minimizes the orthogonal distance from the samples to the line? i.e.,

$$\min_{\mathbf{w}} \sum_{i=1}^{N} d_{\perp}^2(\mathbf{x}_i, \mathbf{w})$$

Note that $\mathbf{x_i} = [x^1, x^2]^T$ is a point and $\mathbf{w}$ is a line in 2D.

Another problem that we are interested in is to find a "representative" for the set of samples that we have. Who should represent the set $\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_N}$?.

How do we define the problem? The problem is to find an entity (such as a point, line, plane etc.) that can represent the set, with minimal loss in "information/content".

8.44 Point that minimizes the distance

Let $\mathbf{z}$ be a point that minimizes the sum of distance to all the given N points. How do we find $\mathbf{z}$?. Let us state the problem as:

$$\min_{\mathbf{z}} \sum_{i=1}^{N} [\mathbf{z} - \mathbf{x_i}]^T [\mathbf{z} - \mathbf{x}_i]$$

Expanding

$$\min_{\mathbf{z}} \sum_{i=1}^{N} \mathbf{z^T z} + \mathbf{x_i^T x_i} - \mathbf{2 z^T x_i}$$

Differentiating with respect to $\mathbf{z}$ and equating to zero:

$$2\mathbf{z} = \sum_{i=1}^{N} 2\mathbf{x_i}$$

or $\mathbf{z}$ is nothing but our familiar mean μ.i.e.,

$$\mathbf{z} = \mu = \frac{1}{N}\sum_{i=1}^{N} \mathbf{x_i}$$

.

This is quite intuitive. Here we only argued that mean minimizes the sum of square distances. Also mean is a good representative of the samples. If we want to represent a set of samples with a single (constant) representation, then it has to be mean.

If we want to represent all the samples with a single dimension (like a new feature, or geometrically like a line), then what it should be? Note that this is also equivalent to finding a new feature that can be used to "approximate" all the d features we have.

We want to look at this new feature as projections on to a line $\mathbf{w}$. as $z_i = \mathbf{w}^T\mathbf{x_i}$. Geometrically, if the structure of the data needs to be preserved (or loss of "information" is small), then we want to minimize the orthogonal distance to the line.

8.45 Minimizing Orthogonal Distance

We are interested in finding (i) a fixed point and (ii) a direction that can define our line. Let $\mathbf{w}$ define the direction. We know the fixed point as mean.

Let us first assume that mean is subtracted from all the samples. Now $\mathbf{x_i}^T\mathbf{x_i}$ is nothing but the square of the distance from the origin, and $\mathbf{w}^T\mathbf{x_i}$ is nothing but the projection of the $\mathbf{x}_i$ on $\mathbf{w}$.

Simple geometry (figure missing) tells us that the orthogonal distance we want to minimze is nothing but

$$\mathbf{x_i}^T\mathbf{x_i} - (\mathbf{w}^T\mathbf{x_i})^2$$

(ref: good old Pythagoras theorem)

Our problem now is

$$\min_{\mathbf{w}} \sum_{i=1}^{N} \mathbf{x_i}^T\mathbf{x_i} - (\mathbf{w}^T\mathbf{x_i})^2$$

First term is independent of $\mathbf{z}$ so we could convert this into a maximization problem as:

$$\max_{\mathbf{w}} \sum_{i=1}^{N} (\mathbf{w}^T \mathbf{x_i})^2$$

or more compactly

$$\max_{\mathbf{w}} \mathbf{w}^T (\mathbf{X}^T \mathbf{X}) \mathbf{w}$$

Here $\mathbf{X}$ is a matrix.

- Q: What is the dimension of $\mathbf{X}$?
- Q: What is the dimension of $\mathbf{X}^T\mathbf{X}$?
- Q: Please refer to the notations that we used for the MSE in the last lecture.

There is an issue here. Unconstrained optimization of this can lead to $\mathbf{w}$ increasing (to infinity). How do we prevent? We need to add a constraint like $\mathbf{w}^T\mathbf{w} = 1$. This means that $\mathbf{w}$ is only a direction and what we want is a unit norm vector that maximizes our objective.

The popular method for introducing the constraint into the optimization problem is with lagrangians. (read more somewhere else.) Popular notation is λ. This modifies the problem as

$$\max_{\mathbf{w}} \mathbf{w}^T (\mathbf{X}^T \mathbf{X}) \mathbf{w} - \lambda(\mathbf{w}^T\mathbf{w} - 1)$$

Now we use our simple trick of differentiating with respect to $\mathbf{w}$ and equating to zero.

$$(X^T X)\mathbf{w} = \lambda \mathbf{w}$$

Or $\mathbf{w}$ is the eigen vector of the $C = \mathbf{X}^T\mathbf{X}$. (Note that C is nothing but our Σ; we just used a different notation so that we will not get confused with the summation Σ) Since the mean was substracted from all the samples, it is easy to see that C is the covariance matrix.

Therefore the line that minimizes the orthogonal distance passes through the origin and has a direction of the first eigen vector of the covariance matrix.

8.45.1 Why first?

It was obvious from the equation that $\mathbf{w}$ will have to be the eigen vector of C. Why did we pick the first eigen vector? Note that C has many more eigen vectors.

- Q: How many non-zero eigen values C can have?
- Q: Let $\mathbf{x}_i \in R^{100}$ and $N = 30$, how many non-zero eigen values $C = \sum_{i=1}^{N} \mathbf{x_i}\mathbf{x_i}^T$ can have?. If $N = 200$, how many it can have?

What is special for the first (largest) or principal one? (in general, it is assumed that eigen values are sorted in non-increasing order.)

We know that $(X^T X)\mathbf{w} = \lambda\mathbf{w}$ and what we want to maximize is $\mathbf{w}^T(\mathbf{X}^T\mathbf{X})\mathbf{w}$. Substituting, we realize that what we want to maximize is $\lambda\mathbf{w}^T\mathbf{w}$. If $\mathbf{w}$ is a unit vector, it is clear that what we want is the eigen vector corresponding to the largest eigen value.

- Q: Can the eigen value be negative or imaginary for C? What are the properties of C?
- Q: Is C is Positive Semi Definite (PSD) matrix? See the definitions and start.

Note: You may also see the notation of Σ for the covariance matrix, at many places.

8.46 Discussions

We now know how to optimize MSE (mean squared error). We also know how to minimize the orthogonal distance and fit a line/plane. In both cases, we formed an appropriate objective function. We minized the objective and found an expression to compute to the optimal solutions in one step.

- This is not always possible. Not all problems are this simple.
- We may not have all the data when we start. We continue to get data online in a streaming manner.
- we do not want to work with "huge" matrices in the casses of large data sets.

This points to the need of incremental techniques to address this class of problem. We will see some of these issues in the next lectures.

8.47 Linear Dimensionality Reduction

Let us come to PCA from a different angle now.

In general, we start with a feature representation $\mathbf{x}$ corresponding to a physical phenomena or an object of interest. (In the case of supervised learning, we also have a corresponding y. i.e., $((\mathbf{x}_i, y_i))$ In practice, this $\mathbf{x}$ is not the result of very careful selection of measurements/features, These features are either what we could think of as relevant or what is feasible in practice. There could be redundancy and correlation within these features too. They may not be the best for our problem

also. A problem of interest to us is to find a new feature representation $\mathbf{z}$, often lower in dimension, as

$$\mathbf{z} = \mathbf{U}\mathbf{x}. \tag{8.11}$$

If $\mathbf{x}$ is a d-dimensional vector and $\mathbf{z}$ is a k-dimensional vector (where $k < d$), $\mathbf{U}$ is a $k \times d$ matrix.

Two variants:

- The above linear transformation (matrix multiplication) leads to a set of "new" features that are "derived" out existing features. New features are linear combination of existing features. We will have a closer look at this today. This can be supervised or unsupervised. i.e., y may be available (supervised) or unavailable (unsupervised).
- Second direction to create a new lower dimensional representation by selecting only the useful/important features from the original set. This is a classical subset selection problem, which is hard to solve. (Similar to your familiar knapsack problem.) This is not in our scope. Such problems are attempted with greedy or backtracking algorithms.

Some people distinguish these two carefully as feature extraction and feature selection. We can also look at them as dimensionality reduction. Dimensionality reduction makes the downstream computations efficient. Some time storage/memory is also made efficient through the dimensionality reduction. If your original $\mathbf{x}$ is "raw-data" (say an image as such represented as a vector), then such techniques are treated as principled ways to define and extract features.

When we do the dimensionality reduction, we may be removing useful information (or noise). Due to this, there is a minor chance that we may reduce the accuracy (read performance) of the down stream task (say classification). In general, dimensionality reduction schemes aim at no major reduction in accuracy (happy with some increase in accuracy), but in a lower dimension.

If "noise" (irrelevant information) is removed or suppressed in the entire process, we may expect some increase in the accuracy. At the same time, if we loose some " relevant information", then we may loose the accuracy. But, alas, we do not know what is noise and what is signal. We would like the machine to figure out this from the examples/data. If both these problems are solved independendly (as is the case in the classical ML schemes), there is noway, we can tell the dimensionality reduction scheme what is the best to do.

The above equation is for linear dimensionality reduction. We also have many equivalent non-linear dimensionality reduction schemes. They are mostly not in our scope.

8.48 PCA

Principal Component Analysis (PCA) is one of the most popular technique for dimensionality reduction. It is unsupervised. It can be seen from two different view points:

- A dimensionality reduction that retains maximum variance along the new dimensions.
- A representation/compression from which one can reconstruct the original data with minimal error.

8.48.1 Minimizing Variance

Let $\mathbf{u}$ be a dimension on which we want to project the data $\mathbf{x}_i$ so that we obtain a new representation z_i that has maximum variance.

It is easy to see that the mean of the original representation gets projected to the mean of the new representation.

$$\bar{z} = \frac{1}{N}\sum_{i=1}^{N} z_i = \frac{1}{N}\sum_{i=1}^{N} \mathbf{u}^T\mathbf{x}_i = \frac{1}{N}\mathbf{u}^T\sum_{i=1}^{N}\mathbf{x}_i = \mathbf{u}^T\mu$$

We are interested in finding a $\mathbf{u}$ that maximizes the variance after the projection

$$\arg\max \frac{1}{N}\sum_{i=1}^{N}(z_i - \bar{z})^2 = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{u}^T\mathbf{x}_i - \mathbf{u}^T\mu)^2$$

$$\arg\max \frac{1}{N}\left(\mathbf{u}^T\sum_{i=1}^{N}[\mathbf{x}_i - \mu][\mathbf{x_i} - \mu]^T\right)\mathbf{u}$$

$$\arg\max \frac{1}{N}\mathbf{u}^T\Sigma\mathbf{u}$$

with the constraint of $||\mathbf{u}|| = 1$, the solution to this problem can be seen as the eigen vector corresponding to the largest eigen value of the covariance matrix Σ. (see the derivation somewhere else in the notes.) When the data is centered or mean is subtracted, one can see that $\Sigma = \mathbf{X}\mathbf{X}^T$. Where $\mathbf{X}$ is a $d \times N$ data matrix. (Note: may be we did use the notation in an earlier lecture for the transpose of this matrix.)

Best dimension to project so as to maximize the variance is the eigen vector corresponding to the largest eigen value. The second best will be the second largest one and so on.

8.48.2 Minimizing Reconstruction Loss

Let $\mathbf{u_1}, \ldots \mathbf{u_d}$ be d orthonormal vectors. We can represent the vectors $\mathbf{x}$ as $\sum_{i=1}^{d}\alpha_i\mathbf{u}_i$. Where the scalar α_i is $\mathbf{x}^T\mathbf{u_i}$. However, if we use smaller than d basis vectors,

there could be some loss or reconstruction error. Let us consider the loss when we use only one $\mathbf{u}$. i.e.,

$$\mathbf{x} - \mathbf{u}\mathbf{u}^T\mathbf{x}$$

Sum of the reconstruction loss for all the N data samples is now:

$$\sum_{i=1}^{N} ||\mathbf{x_i} - \mathbf{u}\mathbf{u}^T\mathbf{x}_i||^2$$

$$= \sum_{i=1}^{N} \left(\mathbf{x}_i^T\mathbf{x}_i + (\mathbf{u}\mathbf{u}^T\mathbf{x}_i)^T(\mathbf{u}\mathbf{u}^T\mathbf{x}_i) - 2\mathbf{x}_i^T\mathbf{u}\mathbf{u}^T\mathbf{x}_i\right)$$

We would like to minimize this. First term is positive (non negative). It is independnet of $\mathbf{u}$. Therefore, we would like to minimize:

$$\sum_{i=1}^{N} (\mathbf{x_i}^T\mathbf{u}\mathbf{u}^T\mathbf{u}\mathbf{u}^T\mathbf{x_i} - 2\mathbf{x}_i^T\mathbf{u}\mathbf{u}^T\mathbf{x}_i)$$

We also know that $\mathbf{u}^T\mathbf{u} = 1$.

$$= \sum_{i=1}^{N} -\mathbf{x}_i^T\mathbf{u}\mathbf{u}^T\mathbf{x}_i = \sum_{i=1}^{N} -\mathbf{u}^T\mathbf{x}_i\mathbf{x}_i^T\mathbf{u} = -\mathbf{u}^T\Sigma\mathbf{u}$$

Minimizing the reconstruction error now becomes that of Maximizing

$$\mathbf{u}^T\Sigma\mathbf{u}$$

with our familiar constraint of $\mathbf{u}^T\mathbf{u} = 1$. This reduces the solution as the eigen vectors corresponding to the largest eigen values.

8.48.3 PCA: Algorithm

- Center the data by subtracting the mean μ
- Compute the covariance matrix $\Sigma = \frac{1}{N}\mathbf{X}\mathbf{X}^T = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i\mathbf{x}_i^T$
- Compute eigen values and eigen vectors of Σ.
- Take the k eigen vectors corresponding to the largest k eigen values. Keep the eigen vectors as the rows and create a matrix $\mathbf{U}$ of $k \times d$.
- Compute the reduced dimensional vectors as

$$\mathbf{z}_i = \mathbf{U}\mathbf{x}_i$$

8.48.3.1 How many eigen vectors?

In practice what should be the value of k? This is often decided by looking at how much information is lost in doing the dimensionality reduction. An estimate of this is

$$\frac{\sum_{i=k+1}^{d}\lambda_i}{\sum_{i=1}^{d}\lambda_i}$$

Often k is picked such that the above ratio is less than 5% or 10%.

Q: Why this ratio is useful? Can you find an explanation?

8.49 Example: Eigen Face

A classical example/application of PCA is in face recognition and face representation. Let us assume that we are given N face images each of $\sqrt{d} \times \sqrt{d}$. We can visualize this as a d dimensional vector. Assume that our input is all the faces from a passport office. All the faces are approximately of the same size. They are all frontal. And also expression neutral.

- Let us assume that mean μ is subtracted from each of the face. If all of the inputs were faces, then the mean is also looking very similar to a face.
- Let the input be $\mathbf{x}_1 \ldots \mathbf{x_N}$ be the mean subtracted faces and $\mathbf{X}$ be the $d \times N$ matrix of all these faces. (Q: Practically which would be larger here? N or d?)
- We are interested in finding the eigen vectors of the matrix $\mathbf{X}\mathbf{X}^T$. Say they are $\mathbf{u_1} \ldots \mathbf{u_k}$. Note that $k \leq \min(N, d)$.
- We then project each of the inputs $\mathbf{x}$ to these new eigen vectors and obtain a new feature.

$$z_i = \mathbf{u_i}^T\mathbf{x} \qquad i = 1, \ldots, k$$

- The new representation $\mathbf{z}$ is obtained like this. You can also look at this as forming a $k \times d$ matrix $\mathbf{U}$ which has its rows eigen vectors.
- We obtain now a set of compact (k dimensional) representation $\mathbf{z_i}$ for each of the input face images $\mathbf{x_i}$, where $i = 1, \ldots, N$

8.49.1 Eigen Vectors of $\mathbf{X}\mathbf{X}^\mathbf{T}$ from $\mathbf{X}^\mathbf{T}\mathbf{X}$

It is worth to see whether N of d is large more closely. In many cases N is larger than d. The reverse is also possible. In the case of eigen faces, it is quite possible that the image size is say 100×100 (or $d = 10000$) and N is only 1000. Let us see how the eigen vectors of $\mathbf{X}^\mathbf{T}\mathbf{X}$ and $\mathbf{X}\mathbf{X}^\mathbf{T}$ are related? The first one is a $N \times N$ matrix while the second is a $d \times d$ matrix. (note that eigen vector computation is a costly numerical computation and a smaller matrix is clearly preferred. Q: What is the computational complexity? $O(?)$)

Let us assume that $\mathbf{u}_1, \ldots \mathbf{u}_m$ are the m eigen vectors of $\mathbf{X}^T\mathbf{X}$ and $\mathbf{v}_1, \ldots, \mathbf{v}_m$ are the eigen vectors of $\mathbf{X}\mathbf{X}^T$. Note that $m \leq \min(d, N)$.

PCA: Proof TODO

there could be some loss or reconstruction error. Let us consider the loss when we use only one $\mathbf{u}$. i.e.,

$$\mathbf{x} = \mathbf{u}\mathbf{u}^T\mathbf{x}$$

Sum of the reconstruction loss for all the N data samples is now:

$$\sum_{i=1}^{N} \|\mathbf{x}_i - \mathbf{u}\mathbf{u}^T\mathbf{x}_i\|^2$$

$$= \sum_{i=1}^{N} (\mathbf{x}_i^T\mathbf{x}_i + (\mathbf{u}\mathbf{u}^T\mathbf{x}_i)^T(\mathbf{u}\mathbf{u}^T\mathbf{x}_i) - 2\mathbf{x}_i^T\mathbf{u}\mathbf{u}^T\mathbf{x}_i)$$

We would like to minimize this. First term is positive (non negative). It is independnet of $\mathbf{u}$. Therefore, we would like to minimize:

$$\sum_{i=1}^{N} (\mathbf{x}_i{}^T\mathbf{u}\mathbf{u}^T\mathbf{u}\mathbf{u}^T\mathbf{x}_i - 2\mathbf{x}_i^T\mathbf{u}\mathbf{u}^T\mathbf{x}_i)$$

We also know that $\mathbf{u}^T\mathbf{u} = 1$.

$$= \sum_{i=1}^{N} -\mathbf{x}_i^T\mathbf{u}\mathbf{u}^T\mathbf{x}_i = \sum_{i=1}^{N} -\mathbf{u}^T\mathbf{x}_i\mathbf{x}_i^T\mathbf{u} = -\mathbf{u}^T\Sigma\mathbf{u}$$

Minimizing the reconstruction error now becomes that of Maximizing

$$\mathbf{u}^T\Sigma\mathbf{u}$$

with our familiar constraint of $\mathbf{u}^T\mathbf{u} = 1$. This reduces the solution as the eigen vectors corresponding to the largest eigen values.

8.48.3 PCA: Algorithm

- Center the data by subtracting the mean μ.
- Compute the covariance matrix $\Sigma = \frac{1}{N}\mathbf{X}\mathbf{X}^T = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i\mathbf{x}_i^T$.
- Compute eigen values and eigen vectors of Σ.
- Take the k eigen vectors corresponding to the largest k eigen values. Keep the eigen vectors as the rows and create a matrix $\mathbf{U}$ of $k \times d$.
- Compute the reduced dimensional vectors as

$$\mathbf{z}_i = \mathbf{U}\mathbf{x}_i$$

8.48.3.1 How many eigen vectors?

In practice what should be the value of k? This is often decided by looking at how much information is lost in doing the dimensionality reduction. An estimate of loss is

$$\frac{\sum_{i=k+1}^{d}\lambda_i}{\sum_{i=1}^{d}\lambda_i}$$

Often k is picked such that the above ratio is less than 5% or 10%.

Q: Why this ratio is useful? Can you find an explanation?

8.49 Example: Eigen Face

A classical example/application of PCA is in face recognition and face representation. Let us assume that we are given N face images each of $\sqrt{d} \times \sqrt{d}$. We can visualize this as a d dimensional vector. Assume that our input is all the faces from a passport office. All the faces are approximately of the same size. They are all frontal. And also expression neutral.

- Let us assume that mean μ is subtracted from each of the face. If all of the inputs were faces, then the mean is also looking very similar to a face.
- Let the input be $\mathbf{x}_1 \ldots \mathbf{x}_N$ be the mean subtracted faces and $\mathbf{X}$ be the $d \times N$ matrix of all these faces. (Q: Practically which would be larger here? N or d?)
- We are interested in finding the eigen vectors of the matrix $\mathbf{X}\mathbf{X}^T$. Say they are $\mathbf{u}_1 \ldots \mathbf{u}_k$. Note that $k \leq \min(N, d)$.
- We then project each of the inputs $\mathbf{x}$ to these new eigen vectors and obtain a new feature:

$$z_i = \mathbf{u}_i{}^T\mathbf{x} \quad i = 1, \ldots, k$$

- The new representation $\mathbf{z}$ is obtained like this. You can also look at this as forming a $k \times d$ matrix $\mathbf{U}$ which has its rows eigen vectors.
- We obtain now a set of compact (k dimensional) representation $\mathbf{z}_i$ for each of the input face images $\mathbf{x}_i$ where $i = 1, \ldots, N$

8.49.1 Eigen Vectors of $\mathbf{X}\mathbf{X}^T$ from $\mathbf{X}^T\mathbf{X}$

It is worth to see whether N of d is large more closely. In many cases N is larger than d. The reverse is also possible. In the case of eigen faces, it is quite possible that the image size is say 100×100 (or $d = 10000$) and N is only 1000. Let us see how the eigen vectors of $\mathbf{X}^T\mathbf{X}$ and $\mathbf{X}\mathbf{X}^T$ are related? The first one is a $N \times N$ matrix while the second is a $d \times d$ matrix. (note that eigen vector computation is a costly numerical computation and a smaller matrix is clearly preferred. Q: What is the computational complexity? $O(?)$)

Let us assume that $\mathbf{u}_1, \ldots, \mathbf{u}_m$ are the m eigen vectors of $\mathbf{X}^T\mathbf{X}$ and $\mathbf{v}_1, \ldots, \mathbf{v}_m$ are the eigen vectors of $\mathbf{X}\mathbf{X}^T$. Note that $m \leq \min(d, N)$.

$$\mathbf{X}^T\mathbf{X}\mathbf{u} = \lambda\mathbf{u} \tag{8.12}$$

$$\mathbf{X}\mathbf{X}^T\mathbf{v} = \lambda\mathbf{v} \tag{8.13}$$

Note that $\mathbf{u}$ is $N \times 1$ and $\mathbf{v}$ is $d \times 1$. Assume $d >> N$. We are interested in computing $\mathbf{v}$ from $\mathbf{u}$.

Let us premultiply both side by $\mathbf{X}^T$.

$$\mathbf{X}^T\mathbf{X}\mathbf{X}^T\mathbf{v} = \lambda\mathbf{X}\mathbf{v}$$

Let us replace $\mathbf{X}^T\mathbf{v}$ by $\mathbf{u}$.

$$\mathbf{X}\mathbf{X}^T\mathbf{u} = \lambda\mathbf{u}$$

The computational procedure can be now:

- Compute the m principal eigen vectors for $\mathbf{X}\mathbf{X}^T$, say $\mathbf{v}_1, \ldots \mathbf{v}_m$.
- Compute the eigen vectors of our interest as

$$\mathbf{u}_i = \mathbf{X}^T\mathbf{v}_i$$

Q: Write the steps of the Eigen face based face representation learning.

CSE 475: Statistical Methods in AI **Monsoon 2019**

SMAI-M-2019 9: LM-III Gradient Descent - I

Lecturer: C. V. Jawahar *Date: DATE*

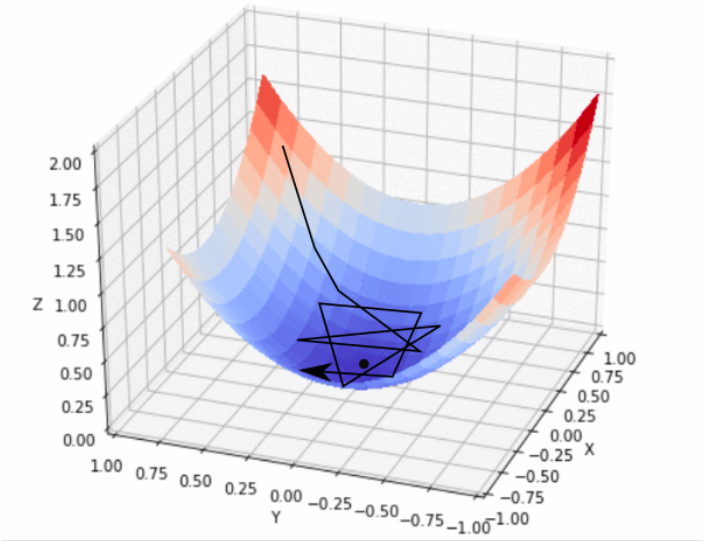
9.50 Challenges with Scale and Optimization

In the last lectures, we had seen how a specific problem of interest can be formulated as an optimization problem. We found closed form solutions to these problems.

However,

1. Such closed form solutions are not always available, specially for most of the optimization problems of our interest.
2. We should also expect many useful modification to such optimization problems in practice, for example, a researcher/practioner may modify the loss function to suite the specific problem requirement. Even if the original problem has a simple closed form solution, this new problem may not have. This handicaps the practical situation.
3. In many practical situations of interest, we need to work with large data sets. This makes the closed form solution, computationally unattractive.
4. More ..

This drives us to look for a simple, yet effective, optimization scheme — popularly known as gradient descent (GD) optimization.



9.51 Gradient Descent

The most popular technique at this stage is gradient descent. i.e.,

- start with a random initialization of the solution.
- incrementally change the solution by moving in the negative gradient of the objective function.
- repeat the previous step until some convergence criteria is met.

Or the key equation for change in weight is:

$$\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k - \eta \nabla J \tag{9.14}$$

Note that $\mathbf{w}$ is a vector. ∇J is a also a vector. Often $\mathbf{w^0}$ is a random vector. For some of the proofs later, we may use $\mathbf{w^0}$ to be zero vector $\mathbf{0}$.

9.51.1 An Intuitive Explanation

Let us look at a simple quadratic error/loss function. (Plot J vs $\mathbf{w}$; Figure missing). Let us assume that $\mathbf{w}$ is a scalar for simplicity.

Let us assume that we start with an arbitrary $\mathbf{w}^0$. How do we want to change $\mathbf{w}$? increase or decrease? (there are only two options for 1D case!!). This is same as negative gradient of the objective. But how much we should increase or decrease? This is the learning rate η. usually a small quantity adhocly set.

- With small learning rate, the iterative algorithm takes more time to converge.
- With large learning rate there is a chance that the algorithm may get diverged or even oscillating.

9.52 Review of Matrix Calculus

Please note that we may be defferentiating scalar valued functions of vectors/matrices at different places. If you are not familiar with this, here is a quick reading:

Thomas Minka, "Old and New Matrix Algebra Useful for Statistics",

`https://tminka.github.io/papers/matrix/minka-matrix.pdf`

Those who are interested in also may read: The Matrix Cookbook

`https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf`

(worth reading, even if we may not read all the results in this course.)

9.53 Specific Examples

9.53.1 Revisiting MSE/Regression

Let us now revisit the MSE we did in the last lecture. Our problem is to

$$\min_{\mathbf{w}} J = \min_{\mathbf{w}} \frac{1}{N} \sum_{i=1}^{N} (y_i - \mathbf{w^T x_i})^2$$

The objective

$$\nabla J = \frac{2}{N} \sum_{i=1}^{N} (y_i - \mathbf{w^T x_i})(-\mathbf{x_i})$$

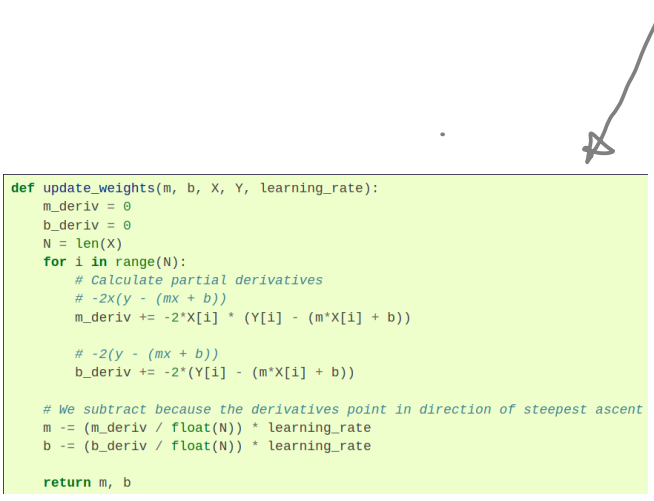
- Q: Write psedocode for gradient descent based MSE; Implement and verify that the solutions of the closed form (last lecture) and this are the same.

This is a well behaved problem (a convex optimization). The solution is simple, and also we reach the same final vector independent of the initialization.

What should be the termination criteria? You can terminate when the changes in the solution is very small (say $< \epsilon$).

9.53.2 Revisiting PCA

- Derive an iterative solution to PCA.
- How do we enfornce the constraint like $||\mathbf{w}|| = 1$?



```python
def update_weights(m, b, X, Y, learning_rate):
    m_deriv = 0
    b_deriv = 0
    N = len(X)
    for i in range(N):
        # Calculate partial derivatives
        # -2x(y - (mx + b))
        m_deriv += -2*X[i] * (Y[i] - (m*X[i] + b))

        # -2(y - (mx + b))
        b_deriv += -2*(Y[i] - (m*X[i] + b))

    # We subtract because the derivatives point in direction of steepest ascent
    m -= (m_deriv / float(N)) * learning_rate
    b -= (b_deriv / float(N)) * learning_rate

    return m, b
```

9.54 Gradient Descent Procedure

9.54.1 Basic Gradient Descent Procedure

We are interested in finding the optimal $\mathbf{w}$ or $\mathbf{w}^*$ corresponding to the minima of $J(\mathbf{w})$. We know the gradient descent optimization procedure for this as:

1. Start with an arbitrary $\mathbf{w}$, $k = 0$
2. Improve $\mathbf{w}$ as
$$\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k - \eta \nabla J$$
3. $k \leftarrow k + 1$
4. Repeat steps 2-3 until some convergence criteria is met.

Convergence criteria can be (i) the change in weight $\mathbf{w}$ or something similar. It is quite intuitive to see that the solution is improving in each iteration.

9.54.2 Practical Issues and Concerns

Gradient descent is a powerful (if not the most powerful at this moment) optimization tool that we will use in many situations in the past.

There are many concerns.

- How do we initialize?
- What learning rate we should choose?
- How do we terminate?

These concerns become more serious when the optimization problem is non-convex. They may not be serious at this stage. There are also many other concerns such as:

- How do we speed up the GD/optimization
- How do we avoid getting trapped in local minima? (or How to find a superior minima)
- Are there better update rules?
- etc.

9.54.3 Stochastic Gradient Descent Procedure

9.54.4 Variations in GD

Single Sample, Vs Batch Vs Mini Batch

Stochastic

SGD with Mini Batch

9.55 Convergence Analysis of GD

1. Convergence of GD does not imply global optima of the loss/objective.
2. Situations where divergence or oscilations can happen.
3. Formal analysis?

CSE 475: Statistical Methods in AI **Monsoon 2019**

SMAI-M-2019 10: LM-IV: Gradient Descent - II

Lecturer: C. V. Jawahar *Date: DATE*

10.57 Gradient Descent - Recap

10.57.1 Concerns

There are many concerns to us at this stage to appreciate this algorithm fully:

- Intuitively, it works, and we appreciate in 1D. What about in a general situations? How do we argue? How did we come up with this update rule?
- There seems to be an adhoc η, learning rate, in the equation. How do we fix this? Indeed, we can find a good η in every iteration. Therefore, it may be worth to write η as η^k, truely.
- What about non-convex functions? That is also a major concern. Surprisingly, this is one of the most favorite methods for optimizing non-convex functions.
- Is this the best update procedure? Can we have better update rules than this?

There are many such very relevant concerns. Let us see if we can find answers to some of these in this lecture.

10.57.2 A trivial case

Consider a function that is linear. (draw $f()$ as a line and x and y are two points on the x axis. Assume we know $f(x)$. How do we compute $f(y)$?

$$f(y) = f(x) + (y - x)f'(x)$$

Simple.!!?

What does it say? If we know xm $f(x)$ and $f'(x)$, then we can compute the function at a new point y. If we want to know where $f(y) = 0$, we can solve $f(x)+(y-x)f'(x) = 0$ and find the y of our interest.

Our problem is not this simple, because the functions of our interest are more complex.

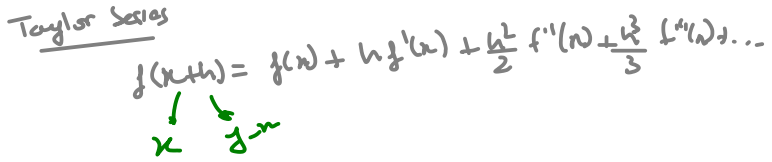
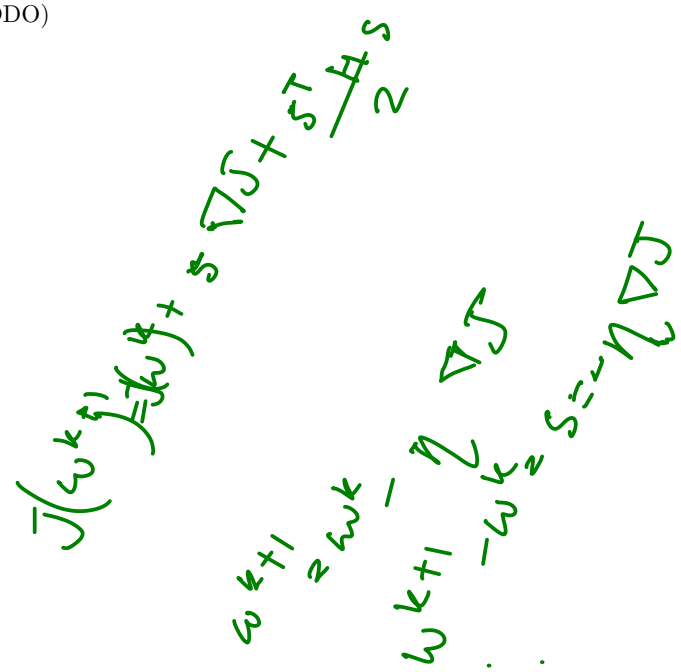


Figure 10.6: A simple figure than explains how GD works! (TODO)



1. Start with an arbitrary $\mathbf{w}$, $k = 0$
2. Improve $\mathbf{w}$ as
$$\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k - \eta \nabla J$$
3. $k \leftarrow k + 1$
4. Repeat steps 2-3 until some convergence criteria is met.

10.58 Taylor Series

We know from the past about a function getting expressed in terms of neighbors. Taylor series is a representation of a function as the sum of terms involving function's derivatives at a single point. It is an infinite series, as long as derivatives do not vanish. You might have studied it in different forms. Hope one of the following equations reminds you the details from your past maths lectures.

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \dots$$

$$f(y) = f(x) + (y-x)f'(x) + \frac{(y-x)^2}{2}f''(x) + \dots$$

$$f(\mathbf{y}) = f(\mathbf{x}) + [\mathbf{y}-\mathbf{x}]\cdot\nabla\mathbf{f}(\mathbf{x}) + \frac{1}{2}[\mathbf{y}-\mathbf{x}]^T\mathbf{H}(\mathbf{x})[\mathbf{y}-\mathbf{x}] + \dots$$

$$\begin{aligned} f(\mathbf{w^{k+1}}) &= f(\mathbf{w^k}) + [\mathbf{w^{k+1}}-\mathbf{w^k}]\cdot\nabla f(\mathbf{w^k}) \\ &\quad + \frac{1}{2}[\mathbf{w^{k+1}}-\mathbf{w^k}]^T\mathbf{H}[\mathbf{w^{k+1}}-\mathbf{w^k}] + \dots \end{aligned}$$

By now, you should realize why Taylor series come at this stage!!

We may also assume our update is more generic and as

$$\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k + \mathbf{s} \tag{10.15}$$

$$f(\mathbf{w^{k+1}}) = f(\mathbf{w^k}) + \mathbf{s}^T\nabla f + \frac{1}{2}\mathbf{s}^T\mathbf{H}\mathbf{s} + \dots$$

Note that we ∇ and Hessians are evaluated at $\mathbf{w}^k$, It is not written here explicitly in some of these equations, to minimize the clutter.

We are now in a position to have a closer look at the GD in the next section. Here our objective function J needs to be expanded and optimized

10.59 Closer Look at GD

Let us first reproduce the two equations that we may need:

$$\begin{aligned} J(\mathbf{w^{k+1}}) &= J(\mathbf{w^k}) + [\mathbf{w^{k+1}}-\mathbf{w^k}]\cdot\nabla J(\mathbf{w^k}) \\ &\quad + \frac{1}{2}[\mathbf{w^{k+1}}-\mathbf{w^k}]^T\mathbf{H}[\mathbf{w^{k+1}}-\mathbf{w^k}] + \dots \end{aligned}$$

$$J(\mathbf{w^{k+1}}) = J(\mathbf{w^k}) + \mathbf{s}^T\nabla J + \frac{1}{2}\mathbf{s}^T\mathbf{H}\mathbf{s} + \dots$$

Let us try to answer three relevant questions:

1. What happens when our familiar gradient descent update rule is used in a general situation?
2. What is the optimal learning rate?
3. Is this the best update rule that we can have?

10.59.1 GD Improves in each step

Let us remember our update rule as

$$\mathbf{w^{k+1}} = w^k - \eta\nabla J(\mathbf{w}^k)$$

or the change in weight (or weight uodate)

$$\mathbf{w}^{k+1} - \mathbf{w}^k = \mathbf{s} = -\eta\nabla J(\mathbf{w}^k)$$

Also let us look at the first order approximation of the Taylor series as:

$$J(\mathbf{w^{k+1}}) = J(\mathbf{w}^k) + \mathbf{s}^T\nabla J(\mathbf{w}^k)$$

Substituting for $\mathbf{s}$

$$J(\mathbf{w^{k+1}}) = J(\mathbf{w}^k) - \eta\nabla J(\mathbf{w}^k)^T\nabla J(\mathbf{w}^k)$$

or

$$J(\mathbf{w^{k+1}}) = J(\mathbf{w}^k) - \eta((\text{a non negative quantity})$$

Therefore, when η is positive, we can argue that the objective is improving in every iteration. When the gradient is zero, no change in the objective and the iterations stops.

One can extend this argument to say that as long as the function is bounded below, we will be able to get to the minima. Depending on η, we may take more iterations. This naturally lead to the question of optimal learning rate.

10.59.2 Optimal learning rate

What is the best learning rate η? Let us reproduce the relevant equations and look at this problem:

$$J(\mathbf{w^{k+1}}) = J(\mathbf{w^k}) + \mathbf{s}^T\nabla J + \frac{1}{2}\mathbf{s}^T\mathbf{H}\mathbf{s}$$

$$\mathbf{w}^{k+1} - \mathbf{w}^k = \mathbf{s} = -\eta\nabla J(\mathbf{w}^k)$$

Substituting

$$J(\mathbf{w^{k+1}}) = J(\mathbf{w^k}) - \eta\nabla\mathbf{J}^\mathbf{T}\nabla\mathbf{J} + \frac{\eta^2}{2}\nabla\mathbf{J}^T\mathbf{H}\nabla\mathbf{J}$$

We want to optimize for η, Let us optimize for η by differentiating with respect to η and equating to zero.

This leads to

$$-||\nabla\mathbf{J}||^2 + \eta\nabla\mathbf{J}^T\mathbf{H}\nabla\mathbf{J} = 0$$

or

$$\eta = \frac{||\nabla\mathbf{J}||^2}{\nabla\mathbf{J}^T\mathbf{H}\nabla\mathbf{J}}$$

Note that this needs change at every iteration k since J and $\mathbf{H}$ are also function of $\mathbf{w^k}$.

This assumes the second order approximation of the function. Though such optimal learning rates could improve the convergence at every point, it also adds an additional computation in each iteration.

10.59.3 Better Update Rule

Let us also see if there is a better update rule. i.e., is there a better $\mathbf{s}$?

Let us revisit:

$$J(\mathbf{w^{k+1}}) = J(\mathbf{w^k}) + \mathbf{s}^T \nabla J + \frac{1}{2}\mathbf{s}^T \mathbf{H}\mathbf{s}$$

Differentiating with respect to $\mathbf{s}$ and equating to zero:

$$\nabla \mathbf{J} + \mathbf{H}\mathbf{s} = 0$$

Or

$$\boxed{\mathbf{s} = -\mathbf{H}^{-1}\nabla \mathbf{J}}$$

This leads to a better update rule known as Newton's updates.

Q: Write the pseudo-code for the newtons method for optimization.

Newton's method is known to provide faster convergence for the optimization. Indeed the second order approximations we use here is more accurrate than the first order one.

Q: However, Newton's method is not very popular in practice. Why is this? You will see plenty of discussions online. Read and appreciate.

10.60 Steepest Gradient Descent

We know from the first order truncated taylor series expansion that

$$J(\mathbf{x} + \Delta\mathbf{x}) = J(\mathbf{x}) + \Delta\mathbf{x}^T \nabla J(\mathbf{x})$$

The directional derivative of ∇J in terms of any direction $\mathbf{u}$ is $\mathbf{u}^T \nabla J$. Our objective is to find what should be the best direction. This leads to the new problem:

$$\min_{\mathbf{u};||\mathbf{u}=1} \mathbf{u}^T \nabla J$$

$$= \min_{\mathbf{u};||\mathbf{u}=1} ||\mathbf{u}|| \cdot ||\nabla J|| \cos\theta$$

This quantity is minized when $\mathbf{u}$ is pointing in the opposite direction of ∇J and $\cos\theta = -1$. Therefore

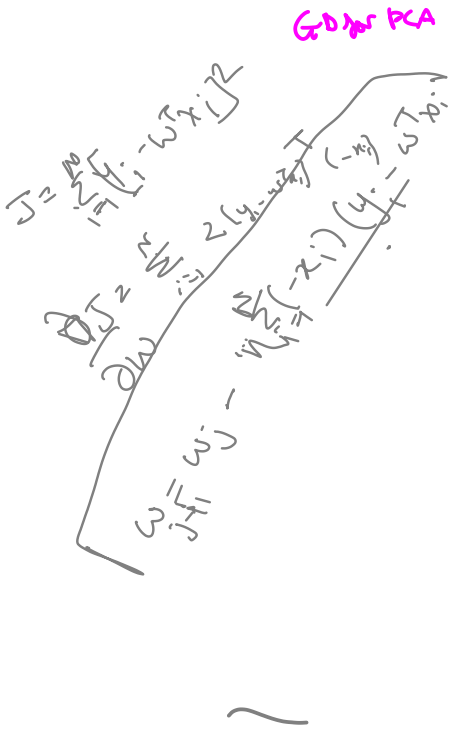
$$\mathbf{x}^{new} = \mathbf{x}^{old} - \eta \nabla J$$

10.61 Discussions

Gradient descent algorithms are the most popular and the most effective ones to train many machine learning models (including deep neural networks). The analysis we had here is quite minimal. Deeper understanding is good to have. If you are interested in this line, here are two references (second one a bit more advanced).

- Leon Bottou, "Stochastic Gradient Descent Tricks, 2012
- Leon Bottou, Frank E. Curtis, Jorge Nocedal "Optimization Methods for Large-Scale Machine Learning", 2018

I advise you to read the first, even if you do not understand everything. Second is for people who can walk that extra mile. Both are beyond the scope of this course, and the exams for sure.



CSE 475: Statistical Methods in AI **Monsoon 2019**

SMAI-M-2019 11: Linear Perceptrons

Lecturer: C. V. Jawahar *Date: 9 Sep 2019*

11.62 GD for Classification

We know the gradient descent equation as

$$\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k - \eta \nabla J \tag{11.16}$$

We start with an initialization $\mathbf{w}^0$ and update the weights/vector until we get the separating hyperplane (or until it converges).

Let us specially focus on the classification problem today. We wish to use a loss that measures classification accuracy. That is, the percentage of samples misclassified.

Let us now consider the objective as

$$J = \frac{1}{N} \sum_{i=1}^{N} (1 - y_i \cdot f(\mathbf{w}, \mathbf{x_i})) \tag{11.17}$$

where $y_i \in \{-1, +1\}$, and $f(\mathbf{x_i}) = sign(\mathbf{w}^T \mathbf{x_i})$. This provides a "unit" loss for all the misclassifications and "zero" loss for all the correct classifications. (see notes elsewhere for details.) We are not using $sign()$. Our ideal goal could have been to minimze this loss.

- Q: Do we have to divide the loss by 2 to get unit loss per sample?
- Q: What makes this problem non-differentiable?

However we have a serious problem. This is not differentiable. Then, what do we do? We can address this in two different ways. This leads to some popular algorithms:

- Perceptron Algorithm
- Logistic Regression
- Support Vector Machines (SVM)

11.63 Perceptron Algorithm

Perceptron algorithm makes the assumption that all the samples are linearly separable. In other words $\exists \mathbf{w}$ such that $y_i \cdot f(\mathbf{w}, \mathbf{x_i}) = +1 \ \forall i$.

- Q: Can every set of (i) Two (ii) Three (iii) Four points be linearly separable in 2D?
- Q: Can any specific set of (i) Two (ii) Three (iii) Four be linearly separable in 2D for all potential labelings of these samples ?
- Q: Can every set of (i) Two (ii) Three (iii) Four points be linearly separable in 2D for all potential labelings of these points?
- *Note: You may know about the Ex-OR case. You may also want to know about VC-Dimension to appreciate the above questions? Why are they important in ML?*

Let us now look at an alternate loss function. We define the loss/objective as

$$J = \sum_{x_i \in \mathcal{E}} -y_i \cdot \mathbf{w}^T \mathbf{x_i} \tag{11.18}$$

where $\mathcal{E}$ is the set of misclassified samples. i.e.,

$$\mathcal{E} = \{\mathbf{x}_i | \mathbf{w}^T \mathbf{x}_i < 0\}$$

Note that J is always non-negative. It can become zero when all samples are correctly classified or $\mathcal{E}$ is empty.

How is this loss function different?

- The summation is only over the misclassified samples. This does not change anything really. (We had zero loss for all the correctly classified ones anyway).
- We then have a "non-unity" loss for all the misclassified ones. (This is different from the previous loss). It is proportional to how far it is from the line/plane/hyperplane.
- Q: Why not then we use the following objective? (Demonstrate with an example)

$$J = \frac{1}{N} \sum_{i=1}^{N} -y_i \cdot \mathbf{w}^T \mathbf{x_i}$$

- In our objective, the farther the sample, the more the loss is. (Indeed, not very ideal!!). *Q: Why do you think this is not ideal? Is it really non ideal if our objective sums only over E*

Pleasantly, this is now differentiable. That is an advantage. Also when all samples are correctly classified (when the problem is linearly separable), the loss becomes zero ($\mathcal{E}$ becomes empty set.) and our algorithm will converge (loss is zero, derivative is also zero.)

There is also an advantage with this loss. A sample that is far from the line/plane will pull/push/rotate the line/plane more than one that is very near the line. This will help in faster convergence.

Let us now re-write our gradient descent equation as:

$$\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k + \eta \sum_{\mathbf{x_i} \in \mathcal{E}} y_i \mathbf{x_i} \tag{11.19}$$

Q: Verify this.

We start with a single sample version of the perceptrin. i.e., the perceptron algorithm is now:

- Initialize $k = 0$, $\mathbf{w}^0$
- While $\mathcal{E}$ is not empty
  - Pick an arbitrary element $\mathbf{x}_j$ from $\mathcal{E}$
  - $\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k + y_j \mathbf{x}_j$
  - $k \leftarrow k + 1$
- Return $\mathbf{w}_k$

Note that (i) We assumed the learning rate $\eta = 1$ (ii) We update the $\mathbf{w}$ for each sample, than for a batch.

For a batch of samples, we can define the weight update as:

$$\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k + \eta \sum_{\mathbf{x}_j \in \mathcal{E}} y_j \mathbf{x}_j$$

Another way we could define the weight update is with the help of desired/target (t) and output (o). In this situation, the update rule is:

$$\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k + \eta \sum_{\mathbf{i=1}}^{N} (t_i - o_i) \mathbf{x_i}$$

- Q: prove that both the update rules are the same (with a scale change in η).

Note that when desired and predicted outputs are same, $t - o$ is zero. Else it is either Positive or negative (when y_i as well as $t - o$ will have the same sign when the sample is misclassified.)

$$\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k + \eta \sum_{i=1}^{N} (t_i - o_i) \mathbf{x_i} \tag{11.20}$$

Note that η is a learning rate and it could absorb any scaling. (η in all these equations need not be identical, see yourself, if they differ by a scale factor.). Here the summation is over all the samples. Note that, this does not change the update rule. The additional terms are zero.

Algorithm now has the following steps:

1. Initialize $\mathbf{w}$, k=0
2. We update the $\mathbf{w}$ as

$$\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k + \eta \sum_{\mathbf{i=1}}^{N} (t_i - o_i) \mathbf{x_i}$$

   or

$$\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k + \eta \sum_{\mathbf{x} \in \mathcal{E}} y_i \mathbf{x_i}$$

3. $k \leftarrow k + 1$
4. Repeat steps 2-4
   - until $\mathcal{E}$ is empty. (the ideal termination criteria for perceptron algorithm) Or
   - until the change in weight is small (say less than θ).

11.64 More Details

11.64.1 Delta Rule

Let us consider a "regression" problem with o_i as $\mathbf{w}^T \mathbf{x}_i$. Consider a least square objective as

$$J = \sum_{i=1}^{N} (t_i - o_i)^2$$

The gradient descent update rule is:

$$\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k + \eta \sum_{\mathbf{i=1}}^{N} (t_i - o_i) \mathbf{x_i}$$

- Q: is it really the same as we saw early? Not really!!

11.64.2 Back to Perceptrons

Is it really possible to arrive the gradient descent update rule as Equation 11.20 from the objective like:

$$J = \sum_{i=1}^{N} (t_i - o_i)^2$$

with $o_i = sign(\mathbf{w}^T\mathbf{x}_i)$? There is a small catch.

With small changes in the $\mathbf{w}$ or the line, the J is not changing. This is not very appropriate to look at this as gradient descent objective. Let us modify this as

$$J = \sum_{i=1}^{N}(t_i - o_i)^2(-\mathbf{w}^T\mathbf{x}_i)$$

This additional terms pulls (or pushes) *proportionally*. Let us now rewrite this J as sum of two parts one over $\mathbf{E}$ and the other on not in $\mathcal{E}$.

$$J = J_1 + J_2 = \sum_{\mathbf{x_i}\in\mathcal{E}}(t_i - o_i)^2(-\mathbf{w}^T\mathbf{x}_i) + \sum_{\mathbf{x}_i not\in\mathcal{E}}(t_i - o_i)^2(-\mathbf{w}^T\mathbf{x}_i)$$

$$J = J_1 + J_2 = \sum_{\mathbf{x_i}\in\mathcal{E}}(2\cdot t_i)^2(-\mathbf{w}^T\mathbf{x}_i) + \sum_{\mathbf{x}_i not\in\mathcal{E}} 0 \times (\mathbf{w}^T\mathbf{x}_i)$$

We know that J_2 is zero. When $\mathbf{x}_i \in \mathcal{E}$, $(t_i - o_i)$ is $2t_i$ i.e., $2y_i$.

$$\nabla J = 2\sum_{i=1}^{N}(t_i - o_i)\mathbf{x}_i$$

Note that:

- Either $(t_i - o_i)$ is zero
- Or we know that $(t_i - o_i)$ is $2t_i$ and $\frac{\partial -\mathbf{w}^T\mathbf{x}_i}{\partial \mathbf{w}} = -\mathbf{x_i}$.

This leads to:

$$\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k - \eta\nabla J$$

$$\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k + \eta\sum_{i=1}^{N}(t_i - o_i)(-\mathbf{x}_i)$$

$$\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k + \eta\sum_{\mathbf{x}\in\mathcal{E}} 2.t_i\mathbf{x}_i$$

$$\mathbf{w}^{k+1} \leftarrow \mathbf{w}^k + \eta\sum_{\mathbf{x}_i\in\mathcal{E}} y_i\mathbf{x}_i$$

(with changes in scale for learning rate η)

11.65 Discussions and Examples

Let us consider some simple situations first and see how the perceptron algorithm behaves.

Let us simplify the update rule first. Assume $\eta = 1$. Let us assume that there is only one sample. Also let us assume that $\mathbf{x}$ has only one dimension i.e., scalar. This simplifies to:

$$w^{k+1} \leftarrow w^k + (t_i - o_i)x_i$$

Now let us consider some situations.

- When there is no misclassification. i.e., $t_i = o_i$. In this case, w does not change.
- Let us consider $t_i = +1$, $o_i = -1$ and x_i is positive. Sample is misclassified. Which means $w^k \cdot x_i$ is negative. (x_i is fixed and positive.) This means w^k is negative. We need to increase w^k to (positive to) minimize/avoid misclassification. If we substitute the values/signs in the above equation, we see that perceptron algorithm does this exactly.
- Now Let us consider $t_i = +1$, $o_i = -1$ and x_i is negative. Sample is misclassified. Which means $w^k \cdot x_i$ is negative. (x_i is fixed and negative.) Therefore, w^k is positive. We need to decrease w^k to minimize/avoid misclassification. If we substitute the values/signs in the above equations, we see that perceptron algorithm does this exactly.
- Q: Convince yourself for all other cases also.

11.65.1 Example/Problem in 2D

Q: Consider a set of vectors in 2D.

$$\{[1,1]^T, [1,3]^T, [2,1]^T, [2,2]^T,$$

$$[-1,-1]^T, [-1,-3]^T, [-2,-1]^T, [-2,-2]^T$$

The first four are from class 1 and the rest four are from class 2.

- Plot the samples in a 2D plane with "$\times$" for positive classes and "$\circ$" for negative class. Is this set linearly separable?
- Start with a random vector for $\mathbf{w}$ and show that the perceptron algorithm converges to a separating line/plane for different values of η. (make sure that you start with a vector that has error!!).
- Why is that the final answers are different for different initializations/η? Then which is the best solution?

11.66 Some Theoretical Results

Perceotron algorithm has a number of interesting theoretical properties/results. A summary is below.

- If there exist a set of weights that are consistent with the data, the perceptron algorithm will converge.
- If the training data is not Linearly Separable, the perceptron algorithm will eventually repeat the same set of weights and thereby enter an infinite loop.

- If the training data is linearly separable, algorithm will converge in a maximum of M steps. (See more below for the bounds).
- Every boolean function can be represented by some network of perceptrons only two levels deep.

11.67 Closer Look at the Theoretical Results

Problem Our problem is to design a classification algorithm for N samples $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots (\mathbf{x}_N, y_N)\}$ where $\mathbf{x}_i \in R^d$ and $y_i \in \{-1, +1\}$

We make the assumption that the data is linearly separable. What does it mean?

This simply means that there exists a solution $\mathbf{w}^* \in R^d$ such that $||\mathbf{w}^*|| = 1$ and a small quantity γ such that

$$y_i(\mathbf{w}^T\mathbf{x}_i) > \gamma$$

for all $i = 1, \ldots, N$. Note that when the $sign(\mathbf{w}^T\mathbf{x})$ is positive and y_i is $+1$, the product is positive. Similarly when the sign $(\mathbf{w}^T\mathbf{x})$ is negative and y_i is 11, the product is still positive. In other-words, when a sample is classified correctly, the product is always positive and the inequality above is true for a small positive quantity γ.

We also make an assumption that samples are bounded. Not a strong assumption. What does it mean?

$$||\mathbf{x}_i|| \leq R \ \forall i$$

11.67.1 Summary of Perceptron Algorithm Assumptions

We start with a random initialization and update solution based on the misclassified samples. We repeat this until we find that all the samples are correctly classified.

We make two assumptions.

1. $y_i(\mathbf{w}^{*T}\mathbf{x_i}) > \gamma$ for all i
2. $||\mathbf{x_i}|| \leq R$ for all i.
3. $||\mathbf{w}^*|| = 1$

Where $\mathbf{w}^*$ is the final/optimal vector. γ is a positive quantity that ensure separability and also a margin. The first one says that the samples are separable. The second one says that all the samples are bounded. (not of infinite magnitude.). Note that γ and R are positive.

11.68 Convergence of Perceptron

Now our main result is *The Perceptron Learning Algorithm makes at most $\frac{R^2}{\gamma^2}$ iterations and converge. It returns a seperating hyperplane after that.*

We know the perceptron update equation as

$$\mathbf{w}^{k+1} = \mathbf{w}^k + y_i\mathbf{x}_i$$

This assumes $\eta = 1$. Also this is a single sample update rule.

Let us assume that we start with $\mathbf{w}^0$ as a zero vector $\mathbf{0}$.

Let us multiply $\mathbf{w}^*$ on both sides.

$$(\mathbf{w}^{k+1})^T\mathbf{w}^* = (\mathbf{w}^k)^T\mathbf{w}^* + y_i(\mathbf{x}_i)^T\mathbf{w}^*$$

since the last term is greater than γ (first assumption).

$$(\mathbf{w}^{k+1})^T\mathbf{w}^* > (\mathbf{w}^k)^T\mathbf{w}^* + \gamma$$

From induction

$$(\mathbf{w}^{k+1})^T\mathbf{w}^* > k\gamma$$

We know that $\mathbf{a}^T\mathbf{b} < ||\mathbf{a}|| \cdot ||\mathbf{b}||$. Therefore,

$$(\mathbf{w}^{k+1})^T\mathbf{w}^* < ||\mathbf{w}^*|| \cdot ||\mathbf{w^{k+1}}|| = ||\mathbf{w^{k+1}}||$$

or finally

$$||\mathbf{w^{k+1}}|| > k\gamma \text{ and } ||\mathbf{w^{k+1}}||^2 > k^2\gamma^2 \tag{11.21}$$

We have now a lower bound on $||\mathbf{w^{k+1}}||^2$

Let us also try to get an upper bound for $||\mathbf{w^{k+1}}||^2$

$$\mathbf{w}^{k+1} = \mathbf{w}^k + y_i\mathbf{x}_i$$

Therefore:

$$||\mathbf{w}^{k+1}||^2 = ||\mathbf{w}^k + y_i\mathbf{x}_i||2$$

$$||\mathbf{w}^{k+1}||^2 = ||\mathbf{w}^k||^2 + ||y_i\mathbf{x}_i||^2 + 2||\mathbf{w^k}^T\mathbf{x_i}||y_i$$

Note that the last term is negative. Since the updates happen only when there is an error in the sample/classification.

$$||\mathbf{w}^{k+1}||^2 \leq ||\mathbf{w}^k||^2 + ||\mathbf{x}_i||^2$$

$$||\mathbf{w}^{k+1}||^2 \leq ||\mathbf{w}^k||^2 + R^2$$

$$||\mathbf{w}^{k+1}||^2 \leq kR^2$$

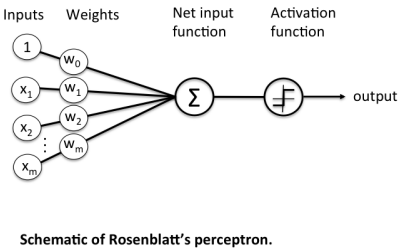


Figure 11.6: A pictorial representation of the neuron. Often a single large circle is shown for a neuron wherein the weighted addition and nonlinearity is combined.

11.68.1 Combining the Bounds

We can now write

$$k^2\gamma^2 < ||\mathbf{w}^{k+1}||^2 \leq kR^2$$

Combing the upper and lower bounds.

$$k < \frac{R^2}{\gamma^2}$$

11.69 Neural Networks View Point

A popular view point of the perceptron is to appreciate it as a "neuron". Though the story has origins in our attempts to understand and reverse engineer human brain and perception skills, it is much easy to appreciate it as a powerful mathematical model at this stage.

A neuron accepts multiple inputs. Weigh each one. Add the inputs. Pass through a nonlinearity. In the case of perceptron, the nonlinearity is a step like nonlinearity. See also figure.

We will revisit the neural network view point at a later stage when we discuss "multi layer perceptrons".

11.70 Variations in Gradient Descent

We now know: (i) How to define a loss function (ii) how to optimize it with gradient descent update equations.

There are in fact many variations in the implementation.

- Single sample or online version: which assumes that samples come one by one. This computes gradient per sample and update for each sample. A sample may be seen multiple times (in a cyclic manner).
- Batch version: Compute the gradient for the batch (full set) and update once.
- Mini-Batch: Instead of taking the full batch, take a smaller batch. This is useful when the data is large.
- Stochastic: Samples in online or mini-batch are selected randomly.

Q: Do write the pseudo code for each of these. Make sure your pseudo code is very close to the programming language that you use.

CSE 475: Statistical Methods in AI **Monsoon 2019**

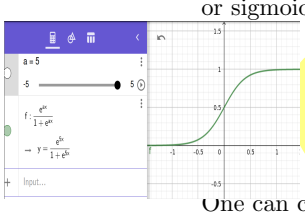
SMAI-M-2019 12: Logistic Regression

Lecturer: C. V. Jawahar *Date: DATE*

- Uses a different function g(a)= sigmoid or tanh.

In the last lecture, we saw that the loss functions that measures simple classification error rates are difficult to optimize. We saw perceptron algorithm as a solution. We will also look at another solution today — Logistic Regression. A very popular solution to the classification problem..

Logistic Regression (LR) is in fact a classification scheme. (Q: Then why is it called regression?) LR introduces an extra nonlinearity $g()$ (called a logistic function or sigmoid) over our familiar $f(\mathbf{w}, \mathbf{x}) = \mathbf{w}^T\mathbf{x}$. i.e.,



$$p(y = 1|\mathbf{x}) = g(\mathbf{w^T x}) = \frac{1}{1 + e^{-\mathbf{w^T x}}}$$

$$p(y = 0|\mathbf{x}) = g(-\mathbf{w^T x}) = \frac{1}{1 + e^{\mathbf{w^T x}}}$$

One can consider this as the posterior probability also.

- Q: Do they sum up to 1? Note that the denominators are not identical. There is a minus sign.
- Q: Plot the $g(\mathbf{w}^T\mathbf{x})$ function against $\mathbf{w^x}$.
- Q: Assume we had a scalar $\alpha \in [0, 1]$ how does the graph of $g(\mathbf{w^T x}) = \frac{1}{1+e^{-\alpha \cdot \mathbf{w^T x}}}$ change ? When does it come and and come close to a step function?

12.71.1 Classification

Let us predict class ω_1 if $f(\mathbf{w}, \mathbf{x}) \geq 0.0$ and ω_2 if $f(\mathbf{w}, \mathbf{x}) < 0.0$. Classification rule corresponding to this is:

$$= \begin{cases} +1 & \text{when } g(\mathbf{w}^T\mathbf{x}) > 0.5 \text{ or } \mathbf{w}^T\mathbf{x} > 0 \\ -1 & \text{when } g(\mathbf{w}^T\mathbf{x}) \leq 0.5 \text{ or } \mathbf{w}^T\mathbf{x} \leq 0 \end{cases}$$

Or else from a probabilistic view, it is to decide to class ω_1 or ω_2 depending on $p(y = 1|\mathbf{x})$ is smaller or larger compared to $p(y = 0|\mathbf{x})$.

$$p(y = 1|\mathbf{x}) = g(\mathbf{w^T x}) = \frac{1}{1 + e^{-\mathbf{w^T x}}}$$

$$p(y = 0|\mathbf{x}) = g(\mathbf{w^T x}) = \frac{1}{1 + e^{\mathbf{w^T x}}}$$

Note that the decision boundary is still $\mathbf{w}^T\mathbf{x} = 0$. Then what changes in logistic regression?

Here, the loss is based on a nonlinear (sigmoid) function.

12.71.2 Closer Look

Let us look at this problem closely. By looking at the sign of $\mathbf{w}^T\mathbf{x}$, we predict the class labels. Let us make a minor change. Let us look for a classifier that predicts the probability of the instance in the class i.e., $p(y|\mathbf{x_i})$. Let us look for a classifier $p(y = 1|\mathbf{x}_i) = \phi(\mathbf{w}, \mathbf{x}_i) = g(\mathbf{w}^T\mathbf{x}_i)$.

$$\phi(\mathbf{w}, \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}}} = \frac{e^{\mathbf{w}^T\mathbf{x}}}{1 + e^{\mathbf{w}^T\mathbf{x}}} \tag{12.22}$$

Note that:

- $0 \leq \phi(\mathbf{w}, \mathbf{x}) \leq 1$
- $p(y = 0|\mathbf{x}; \mathbf{w}) + p(y = 1|\mathbf{x}; \mathbf{w}) = 1$

Note that:

$$1.0 - \frac{1}{1 + e^{-\mathbf{w}^T\mathbf{x}}} = \frac{e^{\mathbf{w}^T\mathbf{x}}}{1 + e^{\mathbf{w}^T\mathbf{x}}} = \frac{1}{1 + e^{\mathbf{w}^T\mathbf{x}}}$$

f(x) = tanh x
f'(x) = 1 −

12.72 Logistic Function

Recollect that our difficulty was directly optimizing an error function which is defined based on the *sign*() or step function. We already saw how perceptrons addressed this. Another way we can handle this is by changing the step function used to a "smooth step function" or a logistic function as

$$g(x) = \frac{1}{1 + e^{-x}} \tag{12.23}$$

This change from zero to one. The transtion from 0 to 1 takes place closer to the origin.

Logistic function is also known sometime as "S" function. (due to the shape) Logistic function, also called sigmoid function is defined as:

$$g(z) = \frac{1}{1 + e^{-z}} = \frac{1}{2} + \frac{1}{2}\tanh(\frac{z}{2})$$

A nice useful property of this function is that its first derivative can be expressed in its own terms:

$$g'(z) = g(z)(1 - g(z))$$

- Q: Do verify this.

12.72.1 Sigmoid and Tanh

Both sigmoid and tanh are two popular nonlinear functions in machine learning. We will see them also as popular activation functions when we talk about neural networks and neuron models.

- In the simplest form, both are smoothed versions of the step functions/discontinuities.
- Sigmoid is in the range $[0, 1]$ while tanh is in the range $[0, 1]$.

$$g(z) = \frac{1}{1 + e^{-z}}$$

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

- Q: Verify $tanh(z) = 2 \cdot g(2z) - 1$

12.73 Loss Function

12.73.1 A Simple Loss Function

One can get tempted to use the same loss function here as:

$$J(\mathbf{w}) = \frac{1}{2N} \sum_{i=1}^{N} (g(\mathbf{w}^T \mathbf{x}_i) - y_i)^2$$

Unfortunately, this is a poor choice of loss function. This is non-convex.

- Question: Prove or Verify that.

Note: we assumed here $y_i \in \{0, 1\}$

12.73.2 MLE based Loss Function

Let us use a cost function from Maximum Likelihood Estimate (MLE) in this case.

Likelihood of the data is given by:

$$l(\mathbf{w}) = \prod_{i=1}^{N} p(y_i | \mathbf{x_i}; \mathbf{w})$$

Looking for $\mathbf{w}$ that maximizes the likelihood

$$\mathbf{w}_{MLE} = \arg\max_{\mathbf{w}} \; l(\mathbf{w}) = \arg\max_{\mathbf{w}} \prod_{i=1}^{N} p(y_i | \mathbf{x_i}; \mathbf{w})$$

(If you are not familiar with the notation, $\prod$ is product, just like $\sum$ is sum.)

The popular trick in formulations like this is to do two transformations of the objective function:

- optimize log of the function instead of the function directly. This converts the multiplication to addition. (note: $\log(ab) = \log(a) + \log(b)$)
- Take negative. It converts the maximization problem to a minimization problem.

12.74 Loss Fn with $y_i \in \{0, 1\}$

Let us look at the loss function with the convention/notation of $y_i \in \{0, 1\}$.

$$p(y = 1 | \mathbf{x}) = g(\mathbf{w}^T \mathbf{x})$$

$$p(y = 0 | \mathbf{x}) = 1 - g(\mathbf{w}^T \mathbf{x})$$

or in a compact manner by combining these two

$$p(y | \mathbf{x}) = g(\mathbf{w}^T \mathbf{x})^y (1 - g(\mathbf{w}^T \mathbf{x}))^{1-y} \tag{12.24}$$

Assuming independence in the data/samples, likelihood is

$$\prod_{i=1}^{N} g(\mathbf{w}^T \mathbf{x_i})^{y_i} (1 - g(\mathbf{w}^T \mathbf{x_i}))^{1-y_i}$$

Taking log, taking negative, we get the objective for the minimization problem as

$$\sum_{i=1}^{N} \left[y_i \log(g(\mathbf{w}^T \mathbf{x_i})) + (1 - y_i) \log(1 - g(\mathbf{w}^T \mathbf{x_i})) \right]$$

12.75 Loss Fn with $y_i \in \{-1, +1\}$

We can also rewrite the objective with the $y \in \{-1, +1\}$ convention.

$$p(y = +1 | \mathbf{x}) = g(\mathbf{w}^T \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^T \mathbf{x}}}$$

$$p(y = -1 | \mathbf{x}) = 1 - g(\mathbf{w}^T \mathbf{x}) = \frac{1}{1 + e^{\mathbf{w}^T \mathbf{x}}}$$

We can combine both into single expression as

$$p(y_i | \mathbf{x_i}) = \frac{1}{1 + e^{-y_i \mathbf{w}^T \mathbf{x}_i}}$$

Assuming independence, the likelihood is

$$\prod_{i=1}^{N} \frac{1}{1 + e^{-y_i \mathbf{w}^T \mathbf{x_i}}}$$

Then the negative log likelihood is

$$\sum_{i=1}^{N} \log(1 + e^{-y_i \mathbf{w}^T \mathbf{x_i}})$$

Explanation of the Objective

- when the sample is classified correctly, $-y_i \mathbf{w}^T \mathbf{x_i}$ is negative and $\log(1 + e^{-y_i \mathbf{w}^T \mathbf{x_i}})$ is nearly zero.
- when the sample is wrongly classified, $-y_i \mathbf{w}^T \mathbf{x_i}$ is positive and $\log(1 + e^{-y_i \mathbf{w}^T \mathbf{x_i}})$ is large.

Question: provide an intuitive explanation why the loss function with $\{0, 1\}$ is also equally good. One can plot the individual loss functions and see that, if $y = 1$, cost is zero if the prediction is correct. if prediction is close to zero, the cost increases to ∞. Similarly one can see for $y = 1$. Larger mistakes get larger penalties.

12.76 Regularization

It is common to regularize the loss function with an additional term.

$$J_R(\mathbf{w}) = J(\mathbf{w}) + \lambda \sum_{i=1}^{d} w_j^2$$

The regularized objective function J_R has an extra term, compared to the original one.

Adding an extra term has many advantages:

- This can avoid overfitting, which is a major concern for us.
- With regularization, we prefer certain type of solutions over other. For example, we encourage, simpler solutions over complex solutions for better "generalization".
- It is common to add an extra term which is the norm of $\mathbf{w}$. If we choose a norm that measures the number of non-zero elements, then while finding the "best" $\mathbf{w}$, we also find one which is sparse. (Q: which norm measures the number of non-zero? Q: How do we minimize such a loss? Read about LASSO and Ridge regression.
- In the GD framework that we use, it is common to add the L2 norm which leads to an additional quadratic term in the objective and then a linear term in the update rule. Since this is an "addition", it does not complicate the derivation/update.

12.77 Gradient Descent Solution

Q: Derive GD update rule and write pseudo code for LR and Regularized LR for (i) single sample , (ii) batch and (iii) mini batch SGD variations. ($3 \times 2 = 6$ algorithms.)

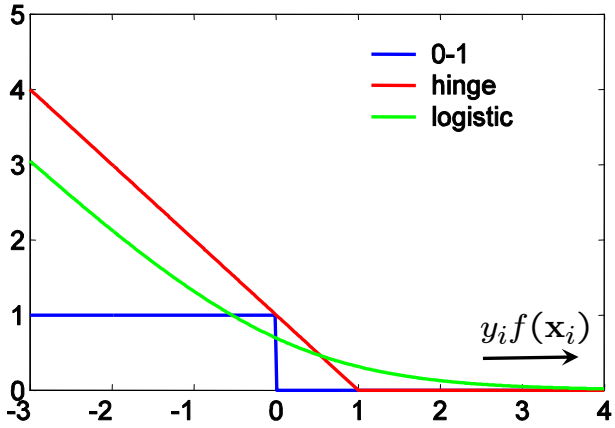


Figure 12.7: Comparison of different losses

12.78 Discussions

Logistic regression is a popular classification scheme. Let us understand the loss in comparison with other methods.

You will see another popular scheme later Support Vector Machine (SVMs). This also optimizes a very similar objective function. Let us write the objective function corresponding to both these schemes in a very similar manner.

SVM:

$$\min_{\mathbf{w}} C \sum_{i=1}^{N} \max(0, 1 - y_i f(\mathbf{x_i})) + ||\mathbf{w}||^2$$

Logistic Regression

$$\min_{\mathbf{w}} \sum_{i=1}^{N} \log(1 + e^{-y_i f(\mathbf{x_i})}) + \lambda ||\mathbf{w}||^2$$

Both objective functions balance the relative importances with an additional term C VS λ. Parameters like this balance the relative importance of terms in an objective function. You may want to guess them right. But not very difficult in most cases.

Let us plot and see how different loss functions look like in the figure 12.7. You can see that both SVM and LR have very similar loss functions. One more smooth than the other.

12.78.1 Margin

In the perceptron algorithm, we had observed that the iterative algorithm terminates when $\mathcal{E}$ is empty. It does not look for a "good" solution; rather it looks for a "valid" solution.

We can see that the MLE based objective also encourage to have a rather large γ (see the notation in perceptron) than a small that perceptron could land up with.

x

12.78.2 Fast and Interpretable

LR is a simple algorithm at the test time. It classifies with $sign(\mathbf{w^x})$. This makes it very efficient at test time, and also scalable.

Far more, w^j tells us the importance of the feature x^j. This also makes it more "interpretable". We can say how much each feature is contributing to the decision or when a decision is made (example a loan is rejected or a person is rejected from entering into a country), we know what feature contributed how much to this decision.

12.79 Multiclass Extension

Most of our discussions till now have been on binary classification. i.e., when the number of classes is 2. However, many practical problems demand more than two classes, say K classes. Can logistic regression be extended for this purpose?

We know that:

$$p(y=1|\mathbf{x};\mathbf{w}) = \frac{e^{\mathbf{w}^T\mathbf{x}}}{1+e^{\mathbf{w}^T\mathbf{x}}}$$

$$p(y=0|\mathbf{x};\mathbf{w}) = 1 - p(y=1|\mathbf{x};\mathbf{w})$$

$$= \frac{1}{1+e^{\mathbf{w}^T\mathbf{x}}} = \frac{e^{-\mathbf{w}^T\mathbf{x}}}{1+e^{-\mathbf{w}^T\mathbf{x}}}$$

Numerator is the weight/confidence of the sample being that class and denominator is the sum of weights for all classes. This allows us to extend to the multiclass setting as

$$p(y=c|\mathbf{x};\mathbf{w_1},\mathbf{w_2},\mathbf{w_K}) = \frac{e^{\mathbf{w_c}^T\mathbf{x}}}{\sum_{i=1}^{K} e^{\mathbf{w_i}^T\mathbf{x}}}$$

Note that the sum of probabilities across all classes sum upto one. Finally a sample is classified into

$$\arg\max_i p(y=i|\mathbf{x})$$

This is also called popularly as **softmax**. Very popular in the modern deep learning architectures.

Q: How does this lead to an objective/loss and a computational procedure for multi-class classification?

$p(1|x) = \frac{e^{w^T x}}{1+e^{w^T x}}$

$p(0|x): 1 - p(1|x)$

$\frac{e^{-w^T x}}{1+e^{-w^T x}}$

$\frac{e^{w_c^T x}}{\sum_{i=1}^{K} e^{w_i^T}}$

CSE 475: Statistical Methods in AI **Monsoon 2019**

SMAI-M-2019 13: More on Linear Methods

Lecturer: C. V. Jawahar *Date: DATE*

13.81 Multiclass Classification

We had seen binary classification schemes. You will see more of them. We know what we need is a multi class classification for many problems. How do we extend these binary classifiers to a multiclass setting in general? We can *fuse* the results of many binary classifiers to obtain a multi class classifier.

Let us consider we have K classes.

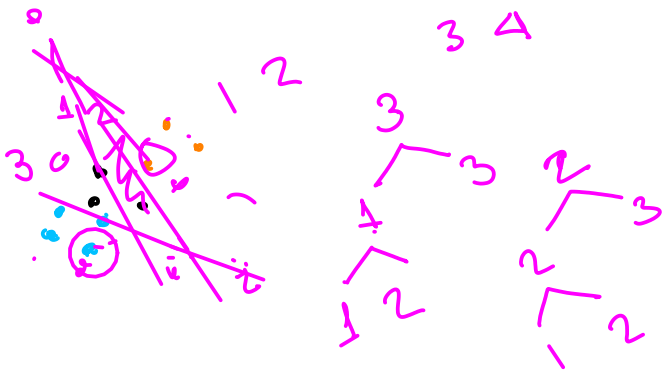
1. We can build K "one Vs rest" classifiers.
2. We can build ${}_KC_2$ pairwise ($ivsj$) classifiers.
3. We can build many (find the combinatorics!) classifier that seperates samples from a set or classes from that of another set of classes.

How do we arrive at a final classification decision in all these cases?

13.81.1 Hierarchical

We achive multiclass classification by hierarchically arranging a number of classifiers of type 3. The advantage is that we can achieve the final class in log() evaluations. However, the challenge is in designing. How many classes need to be trained? How they should be arranged? Popularly this type of classifiers are called Binary Hierarchical Classifier (BHC).

- Q: Suggest a useful heuristics (a greedy algorithm) to design a heirarchical classifier?



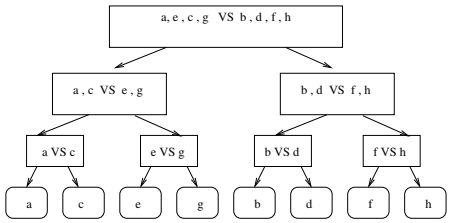


Figure 13.9: Example BHC. How many such BHCs are possible for an 8 class problem?

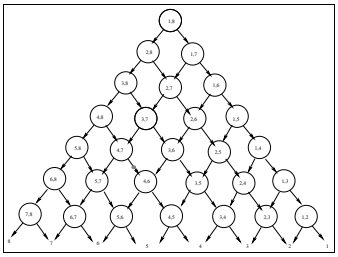
13.81.2 One vs Rest

If the K one vs rest classifiers can provide probabilities, life is simple as we had seen in the case of softmax. Pick the class with highest probability.

If probabilities are not available (and only class label is predicted), then we may have two cases to worry. All classifiers says "rest". or multiple classifiers assign a class-label. In either case, the final decision is difficult without some additional information.

13.81.3 Pairwise Classifiers and Fusion

When there are ${}_KC_2$ classes, one can use simple majority voting to find the best classifier. One can also use DAG (directed acyclic graph) like structure.



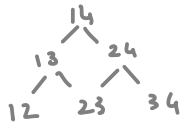
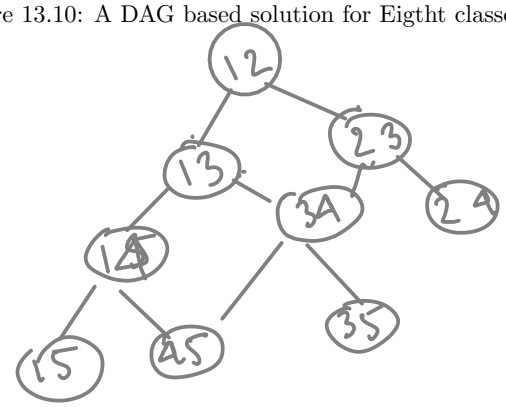


Figure 13.10: A DAG based solution for Eigtht classes



13.82 More on Linear Dimensionality Reduction

We had seen the notion of linear dimensionality reduction

$$\mathbf{z} = \mathbf{W}\mathbf{x}$$

Dimensionality reduction is a vast, important area in the classical machine learning with many implications. Broadly methods are split into:

1. Linear or Non-Linear Methods
2. Local or Global Methods
3. Supervised or Unsupervised Methods

We will not see examples of all these in this course. PCA is an example of (i) Unsupervised (ii) Linear (iii) Global Method.

We will see later, LDA, which is a Supervised dimensionality reduction.

13.82.1 PCA: Summary

- Objective of PCA is to project the data to direction which best *preserves its covariance structure.*
- Let us assume that the data is centered $\sum_i \mathbf{x_i} = 0$. Also $\mathbf{x_i} \in R^N$, $i = 1, \ldots, M$. Centering is done by subtracting the mean from all the samples.
- Covariance matrix $C = \frac{1}{M}\sum_{i=1}^{M} \mathbf{x_i}\mathbf{x_i^T}$ is first calculated. C is a $N \times N$ matrix. We avoid the use of Σ for the covariance matrix here. Let us not get it confused with the $\sum$ we use for Summation later extensively here.
- Assume we have a set of M centered observations: $\mathbf{x_k}, k = 1 \ldots M, \mathbf{x}_k \in \mathbf{R}^N, \sum_{k=1}^{M} \mathbf{x}_k = 0$

  For linear PCA, we solve the equation

  $$C\mathbf{v} = \lambda\mathbf{v}$$

- This leads to eigen vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots$. There are a total of $\min(M, N)$ nonzero eigen values.
- And data is projected to P eigen vectors corresponding to top $P < N$ eigen values of the covariance matrix C.

We see two obvious scopes for improvement:

- PCA does not focus on discriminative nature. It aims at compression/compaction. There is no explicit objective that helps separation.
- PCA is linear. A nonlinear dimensionality reduction could have been more useful.

13.83 LDA

Let us now consider a "discriminative dimensionality reduction". Popularly this is called Linear Discriminant Analysis or Fisher Discriminant Analysis. This is supervised in nature (i.e., we use the class labels y_i also.).

Let us assume that we have two classes A and B. We are interested in finding a new feature $z = \mathbf{u}^T\mathbf{x}$ by projecting on to a new vector $\mathbf{u}$. What are our requirements so that this is good for the classification?

- After the projection, classes should be compact (i.e., samples from A should all come closer and samples from B should all come closer.). All the samples come closer to their own means.
- After the projection classes A and B should be well separated (i.e., they are easily separable.) Means of the classes become farther.

Let us introduce two new notations i.e., within scatter and between scatter of the classes. Both are captured as:

$$S_B = [\mu_1 - \mu_2][\mu_1 - \mu_2]^T$$

$$S_W = S_1 + S_2$$

$$= \sum_{\mathbf{x_i} \in \omega_1} [\mathbf{x_i} - \mu_1][\mathbf{x_i} - \mu_1]^T + \sum_{\mathbf{x_i} \in \omega_2} [\mathbf{x_i} - \mu_2][\mathbf{x_i} - \mu_2]^T$$

Assume we project onto a new (unknown) direction $\mathbf{u}$. After the projection S_B becomes $\mathbf{u}^T\mathbf{S_B}\mathbf{u}$ and S_w becomes $\mathbf{u}^T\mathbf{S_W}\mathbf{u}$.

- Here B indicates "between class"
- and W indicates "within class"

We convert this into a new objective function

$$J(\mathbf{u}) = \frac{\mathbf{u}^T\mathbf{S_B}\mathbf{u}}{\mathbf{u}^T\mathbf{S_W}\mathbf{u}}$$

We would like to maximize this quantity. Since the optima is invariant scaling, let us consider the problem as

$$\max_{\mathbf{u}} \frac{1}{2}\mathbf{u}^T\mathbf{S_B}\mathbf{u}$$

such that $\mathbf{u}^T\mathbf{S_W}\mathbf{u} = 1$. The maximization problem now (cf: Lagrangian)

$$\max_{\mathbf{u}} \frac{1}{2}\mathbf{u}^T\mathbf{S_B}\mathbf{u} - \frac{\lambda}{2}(\mathbf{u}^T\mathbf{S_W}\mathbf{u} - 1)$$

Differentiating with respect to $\mathbf{u}$ and equating to zero.

$$\mathbf{S_B}\mathbf{u} = \lambda\mathbf{S_W}\mathbf{u} \tag{13.25}$$

Such problems are called generalized eigen value problems. $\mathbf{S_B}\mathbf{u}$ is a vector along $[\mu_1 - \mu_2]$. Therefore if $\mathbf{S_W}$ can be inverted,

$$\mathbf{u} = \alpha\mathbf{S_W}^{-1}[\mu_1 - \mu_2]$$

Many questions are left to you to figure out.

- How do we extend this to multi class (beyond two classes)? How does the definition of S_w and S_B change?
- How do we get multiple direction/features/$\mathbf{u}$ than one? What is the second best direction, given that the first one is identified?
- What is the generalized eigen value problem mentioned here? Isn't it LDA also an eigen vector solution?

13.83.1 Tricks

There are a number of tricks that we could do to make our life easy.

1. For example, $\mathbf{S_W}$ could be redefined as:

   $$\mathbf{S_W} = \mathbf{S_W} + \rho\mathbf{I} \tag{13.26}$$

   where ρ is a small real quantity. How does this way of regularizing $\mathbf{S_W}$ help?

2. Alternatively, one could define $\mathbf{S_W}$ as (see "Multiple-Exemplar Discriminant Analysis for Face Recognition" for details). Here the scatter is computed not with respect to mean but with respect to each samples and added.

   $$\mathbf{S_W} = \sum_{i=1}^{C} \frac{1}{N_i^2} \sum_{j=1}^{N_i}\sum_{k=1}^{N_i} [\mathbf{x_j^i} - \mathbf{x_k}^i][\mathbf{x_j^i} - \mathbf{x_k}^i]^T \tag{13.27}$$

   The new definitions of $\mathbf{S_W}$ is some what different from that of the original one. However, both these help in making the $\mathbf{S_W}$ full rank.

3. Another trick is to first apply PCA, and then LDA. This also helps in making $\mathbf{S_W}$ full rank.

Q: *Explain how all the three above methods help in practice?*

13.84 Multi Dimensional Scaling (MDS)

Another popular methods for dimensionality reduction is Multi Dimensional Scaling (MDS).

Like in the previous methods, we can define the dimensionality reduction as a linear transformation:

$$\mathbf{z}_i = \mathbf{W}\mathbf{x}_i$$

Let $\mathbf{D}$ is a $N \times N$ matrix with each element $D(i,j)$ be the distance between $\mathbf{x}_i$ and $\mathbf{x}_j$.

$$D(i,j) = dist(\mathbf{x}_i, \mathbf{x}_j)$$

Let us also look at $\mathbf{D}'beanother$N$\times N$ matrix of distances in the lower dimension.

$$D'(i,j) = dist(\mathbf{z}_i, \mathbf{z}_j)$$

The MDS aims at finding a dimensionality reduction (read $\mathbf{W}$) such that both these distances are as close as possible.

$$\arg\min_{\mathbf{W}} ||\mathbf{D} - \mathbf{D}'||$$

SVM:

Maximize $\frac{w^T w}{2}$

s.t $y(w^T x + b) \geq 1 \ \forall i$

CSE 475: Statistical Methods in AI **Monsoon 2019**

SMAI-M-2019 14: Support Vector Machines

Lecturer: C. V. Jawahar *Date: 26, Sep 2019*

14.86 Maximization of Margin

We now know linear classifiers such as perceptron and logistic regression. We know that if the data is linearly separable, perceptron algorithm will converge with a feasible solution. i..e, a separating hyper plane. However, we know that not all separating hyper planes are equally useful. For example, a plane that "just" classifies a training sample is not the best. Intuitively this leaves higher chance for a test sample (even if it is somewhat similar to the training one) to get misclassified. Conceptually we prefer a separating hyper plane that is far from all the samples.

In short, we want to find a separating hyperplane that maximimzes the margin. That is what support vector machines (SVM) are. SVMs are very popular classifiers even today. They have many nice theoretical properties. The optimization problem is convex and that is a special advantage.

(A figure missing) We know from the mid school mathematics that the distance from origin to the line $ax + by + c = 0$ is $\frac{c}{\sqrt{a^2+b^2}}$. In a similar manner we can see that distance from origin to $\mathbf{w}^T\mathbf{x} + b = 1$ is $\frac{1-b}{||\mathbf{w}||}$. Similarly the distance form origin to the $\mathbf{w}^T\mathbf{x} + b = -1$ is $\frac{-1-b}{||\mathbf{w}||}$. Or the distace between the two side planes is $\frac{2}{||\mathbf{w}||}$.

Thus our objective is to maximize the margin or maximize $\frac{1}{||\mathbf{w}||}$ or minimize $\frac{1}{2}\mathbf{w}^T\mathbf{w}$.

Indeed the unconstrained minimization of this could lead to $\mathbf{w}$ becoming zero. That is not useful. Also this problem does not say anything about the samples correctly classified. We need to add constraints that says that the samples are correctly classified.

- When $y_i = +1$ we would like the samples to be $\mathbf{w}^T\mathbf{x} + b \geq +1$
- When $y_i = -1$ we would like the samples to be $\mathbf{w}^T\mathbf{x} + b \leq -1$.
- We can combine these two constraints into one by multiplying y_i on both side. (Note that when y_i is -1 the inequality sign also reverses. i.e.,

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \quad \forall i$$

14.86.1 Primal Problem

The SVM problem is therefore

$$minimize \ \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

such that

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \quad \forall i$$
$$y_i \in \{-1, +1\}$$

14.87 Solution

The primal problem of interest is

$$minimize \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

such that

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \quad \forall i$$
$$y_i \in \{-1, +1\}$$

This problem can be solved in many ways. We could even use gradient descent to solve this. Being convex in problem, we will obtain the optimal solutions with this. (You could read the paper: Shai Shalev-Shwartz "Pegasos: Primal Estimated sub-GrAdient SOlver for SVM" (though analysis could be hard, initial sections could be OK to read/follow, with some background in optimization. More over, you can write (or download) 20 line matlab or similar code and implement svms!!)

14.87.1 Dual Problem

However, the popular problem is a dual version of the same. (since problem is convex, the optima of the primal and dual will be the same or duality gap will be zero.). With no derivation, let us write the dual problem as

$$maximize \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j \qquad (14.28)$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0$$

with $\alpha_i \geq 0$.

Here α_i are the Lagrangian multipliers. (though popular notation for Lagrangian multipliers is λ, SVMs use α.)

How did we get this dual problem? A brief explanation is given at the end of this lecture.

Dual problem is a classical quadratic programming problem. Many optimization libraries could help in this regard. Let us not aim for writing our own code for this at this stage.

The related to $\mathbf{w}$ is:

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i$$

Q: How do you find b?

14.88 Interpretations

If we had removed some of the training samples, those are away from the side planes, the solution will not change. (why?) The solution depends only on the samples that are hard to classify i.e., the samples on the side planes.

When we solve the dual problem, the αs are sparse. i.e., only some samples have impact on the final solution.

Support Vectors Support vectors are the ones where α_i is non zero.

At the test time, we just need to test the sign of $\mathbf{w}^T\mathbf{x}+b$ and decide whether it is positive or negative class. i.e., decide as

$$sign(\sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b) \tag{14.29}$$

Dot Products Everywhere Another important thing to note is that the samples appear only as dot product in both training (the optimization problem equation 14.28) and the testing (equation 14.29). This is very important and we exploit this smartly when we extend the linear SVMs to nonlinear SVMs using Kernels.

Number of SVs . Assume we do a leave one out testing (LOO). When we leave a non-SV sample and test on it, they all will be correctly classified. Zero error. At the same time if we leave a SVs, and train the solution (i.e., $\mathbf{w}, b$) could change leading to an error. Therefore an upper bound on the error is

$$\frac{\#SV}{N}$$

14.89 Soft Margin SVMs

We made a strong assumption that the samples are linearly separable. That is too restrictive in practice. Let us relax that by allowing a penalty ξ_i is the constraint is violated.

- When $y_i = +1$ we would like the sameples to be $\mathbf{w}^T\mathbf{x} \geq +1 - \xi_i$
- When $y_i = -1$ we would like the sameples to be $\mathbf{w}^T\mathbf{x} \leq -1 + \xi_i$.
- We can combine these two constraints into one by multiplying y_i on both side. (like for hard margin SVM) i.e.,

$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \quad \forall i$$

Indeed if ξ_is are all zero, we will have our hard margin SVM that we saw already. Our new problem of interest is now to minimize both $\mathbf{w}^T\mathbf{w}$ and $\sum_{i=1}^{N} \xi_i$. There are two quantities to simultaneously minimize. We balance the relative importance of these two terms with a non-negative constant C. Our problem is now

$$minimize \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N} \xi_i$$

such that

$$y_i(\mathbf{w}^T\mathbf{x}_i) \geq 1 - \xi_i \quad \forall i$$

If C is too small (say zero), we are easily allowing violations. i.e., the algorithm will look for large margin but discard the concern of violations in the separability. If C is too large, then violations are taken too seriously and not the margin. The parameter C is one that one may have to set in the SVM implementations.

Implementation We know how to implement many of the algorithms that we studied. However, SVMs are not that easy in practice. There are nice implementations like libsvm, liblinear etc. and most of the popular libraries have very good implementations.

Roughly this is what happens:

- Input $(\mathbf{x}_i, y_i)$ for $i = 1, \ldots, N$.
- Solve a quadratic optimization problem, i.e., the dual problem of SVM. Return non-zero α_i or the lagrangians corresponding to the support vectors.
- Given a test sample, compute the class label as $sign(\sum_{i \in SV} \alpha_i y_i \mathbf{x_i}^T \mathbf{x}) + b$.

There are also nice gradient descent solvers for SVMs (read pegasos algorithm, which is also extended for nonliner and softmargin).

14.90 Variations

We already saw the hard margin SVMs and soft margin SVMs. In the first case, we insisted that the sameples should be linearly separable. While in the second we allowed some violations (eg. some outliers or erroneous labels). There are other variations also.

For example the penalty term is L1 or L2 norm of ξ_i.

$$L1 : C\sum_{i=1}^{N}\xi_i$$

$$L2 : C\sum_{i=1}^{N}\xi_i^2$$

(With no technical explanations), minimization of specific norms leads to a solution that is sparse. i.e., we allow some violations but only smaller number of samples are allowed to violate. It is easy to see for $L0$ norm. But that is not what is used in practice.

Q: Derive the dual problem for softmargin L1 and L2 SVMs.

14.91 Primal to Dual

Before we end this lecture, let us also have a quick look how did we arrive at the dual problem from the primal. Some amount of understanding of primal and dual problems in optimization is needed to appreciate this fully (specially to know how the minimization problem became a maximization problem). Here it is more of a simple mathematical exercise of how to rewrite the objectives from primal to dual.

We start with our primal objective of:

Converting the constrained problem to unconstrained problem we have to minimise

$$J(\mathbf{w},b,\alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{N}\alpha_i\left[y_i(\mathbf{w^T x_i}+b)-1\right]$$

where $\alpha_i \geq 0$ are the nonnegative Lagrangian multipliers. The optimality conditions are:

$$\frac{\partial J(\mathbf{w},b,\alpha)}{\partial \mathbf{w}} = \mathbf{0} \text{ and } \frac{\partial J(\mathbf{w},b,\alpha)}{\partial \mathbf{b}} = \mathbf{0}$$

The optimality conditions $\frac{\partial J(\mathbf{w},b,\alpha)}{\partial \mathbf{w}} = \mathbf{0}$ and $\frac{\partial J(\mathbf{w},b,\alpha)}{\partial \mathbf{b}} = \mathbf{0}$ leads to

$$\mathbf{w} = \sum_{i=1}^{N}\alpha_i y_i \mathbf{x}_i$$

$$\sum_{i=1}^{N}\alpha_i y_i = 0$$

To find the optimal values of α which can give the optimal values of $J(\cdot)$,

$$\begin{aligned} J(\mathbf{w},b,\alpha) &= \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{N}\alpha_i\left[y_i(\mathbf{w^T x_i}+b)-1\right] \\ &= \frac{1}{2}\mathbf{w}^T\mathbf{w} - \sum_{i=1}^{N}\alpha_i y_i \mathbf{w^T x_i} - b\sum_{i=1}^{N}\alpha_i y_i + \sum_{i=1}^{N}\alpha_i \end{aligned}$$

The third term of the above objective function is zero and

$$\mathbf{w^T w} = \sum_{i=1}^{N}\alpha_i y_i \mathbf{w^T x_i} = \sum_{i=1}^{N}\sum_{i=1}^{N}\alpha_i\alpha_j y_i y_j \mathbf{x_i^T x_j},$$

Optimal Hyperplane: Solution(Cont.)

The objective function $J_d(\alpha)$ to be maximised becomes

$$J_d(\alpha) = \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{i=1}^{N}\alpha_i\alpha_j y_i y_j \mathbf{x_i^T x_j}$$

Thus find maxima of $J_d(\alpha)$ subject to $\sum_{i=1}^{N}\alpha_i y_i = 0$ and $\alpha_i \geq 0$.

14.91.1 Discussions(*)

Minima of $J(w,b,\alpha)$ is same as Maxima of $J_d(\alpha)$. Why?

A small detour and explanation:**Primal Vs Dual** . You may want to read appropriate material from the optimization literature to appreciate this fully.

Consider a problem of minimizing $f(x)$ such that $\mathbf{g(x)} \geq \mathbf{0}$.

The corresponding lagrangian function is

$$L(\mathbf{x},\lambda) = f(\mathbf{x}) - \lambda^{\mathbf{T}}\mathbf{g(x)}$$

Now,

$$\max_{\lambda\geq 0} L(\mathbf{x},\lambda) = \begin{cases} \infty & \text{if } g(x) < 0 \\ f(x) & \text{otherwise} \end{cases}$$

Primal Problem: $\min_x \max_{\lambda\geq\mathbf{0}} L(\mathbf{x},\lambda)$

Dual Problem: $\max_{\lambda\geq\mathbf{0}} \min_x L(\mathbf{x},\lambda)$

A primal problem of minimization over x became a maximization problem over the Lagrangians λ.

a detailed intro to primal and dual problems in optimization is beyond the scope of this course. Please read on Internet, if interested in more details.

$\min \frac{1}{2}w^Tw$ s.t. $y_i(w^Tx_i+b) = 1$

Lagrangian Formulation

- So in the SVM problem the Lagrangian is

$$\min L_P = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{l} a_i y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b\right) + \sum_{i=1}^{l} a_i$$

s.t. $\forall i, a_i \geq 0$ where l is the # of training points

- From the property that the derivatives at min = 0 we get:

$$\frac{\partial L_P}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{l} a_i y_i \mathbf{x}_i = 0$$

$$\frac{\partial L_P}{\partial b} = \sum_{i=1}^{l} a_i y_i = 0 \text{ so}$$

$$\mathbf{w} = \sum_{i=1}^{l} a_i y_i \mathbf{x}_i, \quad \sum_{i=1}^{l} a_i y_i = 0$$

Primal problem:

$$\min L_P = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{l} a_i y_i \left(\mathbf{x}_i \cdot \mathbf{w} + b\right) + \sum_{i=1}^{l} a_i$$

s.t. $\forall i$ $a_i \geq 0$

$$\mathbf{w} = \sum_{i=1}^{l} a_i y_i \mathbf{x}_i, \quad \sum_{i=1}^{l} a_i y_i = 0$$

Dual problem:

$$\max L_D(a_i) = \sum_{i=1}^{l} a_i - \frac{1}{2}\sum_{i=1}^{l} a_i a_j y_i y_j \left(\mathbf{x}_i \cdot \mathbf{x}_j\right)$$

s.t. $\sum_{i=1}^{l} a_i y_i = 0$ & $a_i \geq 0$

(note that we have removed the dependence on **w** and b)

Non-linear SVMs

So, the function we end up optimizing is:
$L_d = \Sigma a_i - \frac{1}{2}\Sigma a_i a_j y_i y_j K(x_i \bullet x_j)$,

Kernel example: The polynomial kernel
$K(xi,xj) = (x_i \bullet x_j + 1)^p$, where p is a tunable parameter
Note: Evaluating K only requires one addition and one exponentiation more than the original dot product

CSE 475: Statistical Methods in AI **Monsoon 2019**

SMAI-M-2019 15: Nonlinear SVM and Kernels

Lecturer: C. V. Jawahar *Date: Sep 30, 2019*

15.93 Kernels and Feature Maps

We had seen the idea of dimensionality reduction in the past. We had seen PCA. We can also have many other dimensionality transformation techniques. The idea is that with an appropriate feature transformation the problem becomes "simpler" and the classifier/algorithm can do superior. Why don't we increase the dimension if that makes the problem simpler?

Consider a feature transformation (or feature map) as:

$$\mathbf{p} \rightarrow \phi(\mathbf{p}).$$

Let us start with a specific example: Let $\mathbf{p} = [p_1, p_2]^T$, and $\phi(\mathbf{p})$ be $[p_1^2, p_2^2, \sqrt{2}p_1p_2]$. You may wonder how this helps us or why we do this. Wait. We will see the utility as we move forward. Note the notations. Here the sample $\mathbf{p}$ has two features/dimensions p_1 and p_2.

A number of algorithms in machine learning use dot product as the basic operation. You already had seen $\mathbf{w}^T\mathbf{x}$. The beauty of dot product is that the result is scalar independent of the dimensionality of the vector. i.e., the dot product of two vectors in 2D and 3D will be still a scalar, independent of the dimension. Let us now compute the dot product of $\phi(\mathbf{p})$ and $\phi(\mathbf{q})$. Here $\mathbf{q}$ is also a 2D vector similar to $\mathbf{p}$. i.e., $\mathbf{q} = [q_1, q_2]^T$. We know that $\mathbf{p}^T\mathbf{q} = p_1q_1 + p_2q_2$. Let us compute the dot/scalar/inner product in the new feature space.

$$\begin{aligned}\phi(\mathbf{p})^T\phi(\mathbf{q}) &= [p_1^2, p_2^2, \sqrt{2}p_1p_2]^T[q_1^2, q_2^2, \sqrt{2}q_1q_2] \\ &= p_1^2q_1^2 + p_2^2q_2^2 + 2p_1p_2q_1q_2 \\ &= (p_1q_1 + p_2q_2)^2 \\ &= (\mathbf{p}^T\mathbf{q})^2 = \kappa(\mathbf{p}, \mathbf{q})\end{aligned}$$

What it says is something simple. If we want to compute the dot products of $\phi(\mathbf{p})$ and $\phi(\mathbf{q})$, what we need to do is only computing dot product of $\mathbf{p}$ and $\mathbf{q}$ and then square it. (indeed, that is true only for this specific $\phi()$) Note that if we compute $\phi(\mathbf{p})^T\phi(\mathbf{q})$ like this, we do not have to really compute $\phi()$ explicitly. That could be a huge advantage in many situations. Later today, we also would like to map $\mathbf{p}$ into infinite dimension with a $\phi()$. The advantage of not requiring to compute $\phi()$ will be a big advantage then.

Feature Map: Here, $\phi()$ is popularly called as a feature map.

Kernels: The function $\kappa()$ is a kernel function.

We saw the kernel function of $(\mathbf{p}^T\mathbf{q})^2$. This need not be square. It could be $(\mathbf{p}^T\mathbf{q})^d$. This is a polynomial kernel. With some effort we can write the $\phi()$ corresponding to this kernel. There are many other potential kernels. A kernel functions allows us to compute an inner/dot product in a new feature space.

Let us consider another $\phi()$ for the same $\mathbf{p} = [p_1, p_2]^T$. Let $\phi(\mathbf{p})$ as $[p_1^2, p_2^2, p_1p_2, p_2p_1]$. Here $\phi()$ maps from 2D to 4D. (instead of 2 to 3 as in the previous example).

$$\begin{aligned}\phi(\mathbf{p})^T\phi(\mathbf{q}) &= [p_1^2, p_2^2, p_1p_2, p_1p_2]^T[q_1^2, q_2^2, q_1q_2, q_2q_1] \\ &= p_1^2q_1^2 + p_2^2q_2^2 + p_1p_2q_1q_2 + p_1p_2q_1q_2 \\ &= (p_1q_1 + p_2q_2)^2 \\ &= (\mathbf{p}^T\mathbf{q})^2 = \kappa(\mathbf{p}, \mathbf{q})\end{aligned}$$

There can be many such kernels functions and feature maps. The above ones were only some examples. Does it mean that for any $\kappa(,)$ we have a corresponding $\phi()$? Can any function be a kernel function? There are many such curious questions for investigating later.

Q: What about a kernel like $(\mathbf{p}^T\mathbf{q})^2 + (\mathbf{p}^T\mathbf{q})^3$? What will be the corresponding feature map?

15.93.1 Motivation from Separability

Consider a 2D pattern of two concentric circles (figure missing). Inner circle is class 1 and outer circle from class 2. Consider a feature map of the form

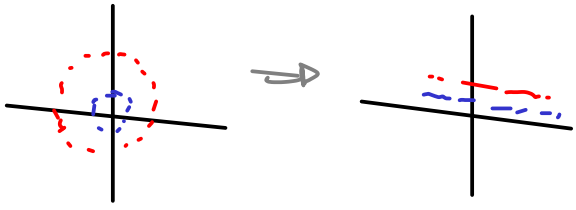
$$\phi(\mathbf{p}) = \phi([p_1, p_2]^T) = [r, \theta]^T$$

where (r, θ) is the polar coordinates. It is simple to see that the concentric circles will become separable in the polar coordinates. (indeed only r could have been enough.)

We can also consider

$$\phi(\mathbf{p}) = \phi([p_1, p_2]^T) = [p_1^2, p_2^2]^T$$

What happens to the concentric circles?



The point to note is that a 'good' $\phi()$ is going to make our (classification) problem simpler (say linearly separable). There are more unanswered questions now. How do we find the useful $\phi()$ or $\kappa()$ for a new problem.

15.94 Kernel Matrix

For many problems we have N sample vectors and we need to compute all kernel values for all pairs. Popularly, a kernel matrix $\mathbf{K}$ with (i, j) th element as $\kappa(\mathbf{x}_i, \mathbf{x}_j)$.

Q: What are the properties of the kernel matrix? square? symmetric? what more?

15.95 Nonlinear/Kernel SVMs

The notion of Kernels is very nicely related to SVMs. One may wonder whether SVMs are designed for Kernels or Kernels are designed for SVMs!!.

Let us now come back to our SVM problem (dual).

$$\max \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \tag{15.30}$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0$$

with $\alpha_i \geq 0$. Solving this gives us a linear SVM.

Assume the input data $\{(\mathbf{x}_i, y_i)\}$ was mapped by $\phi()$, leading to $\{(\phi(\mathbf{x}_i), y_i)\}$. We can find a linear boundary in the new feature space. And the corresponding decision boundary in the original space is going to be a nonlinear one.

This makes the SVM problem (with appropriate constraints) as

$$\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

or

$$\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j) \tag{15.31}$$

Many practical solvers use the precomputed $N \times N$ kernel matrix. This avoids repeated computation of kernel functions. But this necessitates the need to store and manipulate a $N \times N$ matrix while solving. Not very nice for big data sets.

15.95.1 Training and Testing

When you solve this nonlinear SVM problem, we get α_i corresponding to the new nonlinear SVM. Or what we get is the nonlinear SVM in the original feature space. Typically the output of the training is a set of α_i (that are non zero).

At the test time, we only need to evaluate the sign of $\mathbf{w}^T \phi(\mathbf{x}) + b$. i.e.,

$$sign(\sum_{i=1}^{N} \alpha_i y_i \phi(\mathbf{x}_i)^T \phi(\mathbf{x}) + b) \tag{15.32}$$

In a kernel setting this simply becomes:

$$sign(\sum_{i=1}^{N} \alpha_i y_i \kappa(\mathbf{x}_i, \mathbf{x}) + b) \tag{15.33}$$

Please note that α_i is sparse and we need to sum this only over the support vectors. However, at the test time, we need to have all the support vectors with us and the number of kernel evaluations is related to the number of support vectors. This makes the nonlinear SVMs slower at run time.

In the case of linear SVM, it is possible to compute $\mathbf{w}$ upfront and make each of the testing simple in computation. (Indeed while training, we still need to solve for α_i.).

Will the support vectors (and the magnitude of α_i) change with the choice of kernels? Yes. If you change the kernel, you need to solve the problem again and obtain the new α_i.

15.95.2 Example

Consider an example of XOR problem with $(-1, -1), -1; (-1, +1), +1; (+1, -1), +1; (+1, +1), -1$
Clearly the given four samples are not linearly separable. Assume we use a kernel $\phi(\mathbf{p}) = p_1 p_2$, the data becomes immediately separable. $+1, -1; -1, +1; -1, +1; +1 - 1$.

A quadratic kernel like $\kappa(\mathbf{p}, \mathbf{q}) = (\mathbf{p}^T \mathbf{q}))^2$ or $(\mathbf{p}^T \mathbf{q}+1)^2$ will have a product term and this can make the XOR problem separable. (remember the example $\phi()$ we had at the beginning.)

15.96 Popular Kernels

We appreciate

- the utility of feature maps and kernels in"simplifying" the problem (eg. making the problem linearly separable).

- the fact that we do not have to evaluate the feature map $\phi()$ in practice. A small kernel computation in the original space is equivalent to the inner product in the feature space. An important point to note is that when we use kernels, we do not evaluate $\phi()$ or even do not have to know the explicit form of $\phi()$ itself. (Knowing $\phi()$ is still required for the exams!!)

The feature maps and the kernels that we saw is a toy kernel in 2D. We saw the $\phi()$ mapping frm 2D to 3D or 4D. Why not $\phi()$ mapping to an infinite dimensional space? Many popular kernels do that.!!

Many of the popular kernel functions

Linear Kernel	: $K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})$
Polynomial Kernel	: $K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^d$
Radial Basis Kernel (RBF)	: $K(\mathbf{x}, \mathbf{y}) = e^{-\gamma \|\mathbf{x}-\mathbf{y}\|^d}$
Sigmoid Kernel	: $K(\mathbf{x}, \mathbf{y}) = tanh(\gamma(\mathbf{x} \cdot \mathbf{y}))$

We may not know the "right" kernel for a problem in our hand always. For example a simple product term worked for XOR. But a polynomial kernel, which has such a term is what we may use in practice. In practice we try multiple popular kernels (i.e., the ones that are in your library!!) and pick the one that gives us best results. RBF kernel is often preferred for many problems.

15.96.1 Kernels for more structured data

We can think of kernels as "similarity functions" that can be plugged into the machine learning algorithm. We had seen kernels for real vectors. In many problems the inputs could be objects like (i) strings, (ii) trees or (iii) graphs. Can kernels be used in such situations?

One can define kernel over strings, trees, graphs etc. There are many such kernels available in the literature. This makes it possible to treat objects like graphs the same way we do the vectors in vector spaces.

Now we can directly work on strings in SVMs. An SVM can train and test Strings (not just vectors) given that the appropriate kernels are used. This also makes the SVM more general and useful.

15.97 Discussions

15.97.1 What is a valid kernel?

We know a number of functions like $(\mathbf{p}^T\mathbf{q})^d$ and $(\mathbf{p}^T\mathbf{q} + 1)^d$ as polynomial kernels.

We also saw a number of exponential kernels. Many of them also correspond to infinite dimensional feature maps. Remember we do not have to find the feature maps always.

Will every kernels function have a corresponding feature map? or what is a valid kernel?

When Kernel matrix K is PSD, the corresponding kernel is valid. This comes from the Mercer's theorem.

15.98 Beyond this course

15.98.1 What more? (*)

Kernels is an exciting topic. There has been extensive use of the "kernel trick" in converting a wide range of linear algorithms into nonlinear form. This takes advantage of the numerical stability of linear algorithms and expressive power of nonlinearity. This resulted in algorithms like Kernel Perceptron, Kernel PCA, Kernel LDA etc. Here is an additional reading. (not an original exposition; not very recent)

- https://www.dropbox.com/s/qryziuo3u143q5e/KERNEL-REVIEW.pdf?dl=0

Urge you to do a quick reading/glance to get better idea of this space.

15.98.2 Kernels for Structured Data (*)

Kernels can be defined for many structured data like strings, trees and graphs. This area has many interesting ideas. Here is a quick read:

`https://people.eecs.berkeley.edu/~jordan/kernels/0521813972c11_p344-396.pdf`

15.98.3 Can Kernels be learned? (*)

A natural question with which we end this lecture is "how do I find the right kernel for my problem?". In practice, most people do not worry about this problem and use a "powerful" kernel like RBF kernel and move on.

But there are many attempts to find the right kernel for a problem. Assume we define the optimal kernel as a linear combination of a number of base kernels, we can define this problem as that of learning the coefficients (say β_i) of this linear combination.

$$\kappa(\cdot, \cdot) = \sum_{i=1}^{P} \beta_i \kappa_i(\cdot, \cdot)$$

There have been some nice algorithms and attempts in such learning, popularly known as Multiple Kernel Learning.

CSE 475: Statistical Methods in AI **Monsoon 2019**

SMAI-M-2019 16: Kernelization

Lecturer: C. V. Jawahar *Date: 7, Oct 2019*

16.99 Kernels and Kernelization

We know Kernels as a powerful tool to bring in "non-linearility" into the linear algorithms. With the help of kernels, we combine:

- the expressive power due to the non-linearity
- the elegance and robustness of the linear methods.

This led to bringing kernels into a wide variety of classical/popular linear algorithms, in the past. Today we see two such examples.

Objective of this lecture is to appreciate the notion of kernels, beyond SVMs.

16.100 Kernel Perceptron

Basic Perceptron

We know the perceotron algorithm from the past lectures, as a simple algorithm that can learn a separating hyperplane (linear function). Let us revisit this algorithm today, and see how to "kernalize" this algorithm.

Ver 1 We start with a version of the perceptron algorithm, that we know. It is possibly written down in a form that helps us in kernalizing. Here we assume that the weight vector is initialized as zero vector $\mathbf{0}$. Also, we assume that the learning rate is 1.0. We assume that the samples are linearly seperable.

Note that the representation $< \mathbf{w}, \mathbf{x_i} >$ is same as $\mathbf{w}^T\mathbf{x}_i$

Algorithm 1 PerceptronPrimal($\{(\mathbf{x}_i, y_i)\}$ for $i = 1, 2, \cdots, N$)

Require: $\exists \mathbf{w} \ni y_i = sign(\langle \mathbf{w}, \mathbf{x}_i \rangle)$ for $i = 1, 2, \cdots, N$

```
w = 0
repeat
    for i = 1 to N do
        if y_i ⟨w, x_i⟩ ≤ 0 then
            w ← w + y_i x_i
        end if
    end for
until no sample is misclassified
```

Figure 16.11: Our familiar perceptron algorithm

We can make a simple observation here. The final weight vector is a linear combination of the training data.

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i \mathbf{x}_i$$

Note that we started with $\mathbf{w}$ as zero and it was changed only as $\mathbf{w} \leftarrow \mathbf{w} + y_i\mathbf{x}_i$.

What is α_i here? It is related to the number of times the sample was used to modify the solution. (or number of times it entered the inside the condition).

- Q: What does it mean if α_i is zero?
- Q: Is α_i always non-negative?
- Q: Where is y_i absorbed in the final equation?
- Q: If we had a learning rate η in the original learning algorithm, does it change anything? Does the assumption of $\eta = 1.0$ constraining in any form?

Ver 2 If we appreciate the fact that $\mathbf{w} = \sum_i \alpha_i \mathbf{x}_i$, we can re-write $\mathbf{w}^T\mathbf{x}_j$ as

$$\sum_{i=1}^{N} \alpha_i \mathbf{x}_i^T \mathbf{x}_j = \sum_{i=1}^{N} \alpha_i G_{ij}$$

where G is Gram matrix, with elements defined as $G_{ij} = \mathbf{x}_i^T\mathbf{x}_j$

Algorithm 2 PerceptronDual($\{(\mathbf{x}_i, y_i)\}$ for $i = 1, 2, \cdots, N$)

Require: $\exists \alpha_j \ni y_i = sign(\langle \sum_{j=1}^{N} \alpha_j \mathbf{x}_j, \mathbf{x}_i \rangle)$ for $i, j \in \{1, 2, \cdots, N\}$

```
α_i = 0 for i = 1, 2, ··· , N
compute the Gram Matrix G_ij = ⟨x_i, x_j⟩
repeat
    for i = 1 to N do
        if y_i Σ_{j=1}^{N} α_j G_ij ≤ 0 then
            α_i ← α_i + y_i
        end if
    end for
until no sample is misclassified
```

Figure 16.12: A variant of the perceptron algorithm. Note the equivalence

Kernel Perceptron

We have now seen a linear algorithm that uses only dot products on the input data. This is the right time to think of Kernalizatin!!.

Ver 3: Kernel Perceptron Now that we have expressed the perceptron algorithm in terms of dot products, we can re-write the perceptron algorithm as "Kernel perceptron" with Gram matrix replaced by a Kernel matrix K, with (i, j) element defines as:

$$K_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$$

Algorithm 3 KernelPerceptron($\{(\mathbf{x}_i, y_i)\}$ for $i = 1, 2, \cdots, N$, $\kappa(.,.)$)

$\alpha_i = 0$ for $i = 1, 2, \cdots, N$
compute the Kernel Matrix $\mathbf{K}_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$
repeat
  for $i = 1$ to N **do**
    if $y_i \sum_{j=1}^{N} \alpha_j \mathbf{K}_{ij} \leq 0$ **then**
      $\alpha_i \leftarrow \alpha_i + y_i$
    end if
  end for
until no sample is misclassified

Figure 16.13: Kernel Perceptron Algorithm

Inference/Testing How do we evaluate the class/label of a new sample $\mathbf{x}$? In the classical case, we did it as

$$sign(\mathbf{w}^T \mathbf{x}).$$

In the kernelized version, we can do it as

$$sign(\sum_{i=1}^{N} \alpha_i \kappa(\mathbf{x}_i, \mathbf{x}))$$

Discussions

- Can K-Perceptron be used for linearly non-separable data?
- What are the computational advantages/disadvantages of this algorithm?

16.101 Kernel PCA

Let us now kernalize another of our familiar algorithm – PCA. The trick for kernalizing the PCA algorithm is some what different from that of the perceptron algorithm. (It is not just re-writing the pseudo-codes!)

We make a simple observation that eigen vectors can be expressed as a linear combination of the input samples. This makes the PCA algorithm Kernalization friendly. Also note that we do not need the eigen vectors. We only need the dot product of any sample with the eigen vectors.

Notation: We use the following notations:

- M is the number of samples. (should have been N itself)
- C is the covariance matrix. We did not use Σ to avoid any potential confusion with the symbol for summation ($\sum$).
- $\mathbf{v}$ (and $\mathbf{V}$) is the eigen vector (EV).

16.101.1 Preliminaries

We start with the assumption that the data is centered, or the mean of the data is zero. This is not very restrictive. We also remove it at a later stage. Then we know that the covariance matrix is:

$$C = \frac{1}{M} \sum_{i=1}^{M} \mathbf{x_i} \mathbf{x_i^T} \text{ and } C\mathbf{v} = \lambda \mathbf{v}$$

EV is a linear combination of samples We know that $C\mathbf{v} = \lambda \mathbf{v}$. Or else,

$$\mathbf{v} = \frac{1}{\lambda M} \sum_{i=1}^{M} \mathbf{x_i} \mathbf{x_i^T} \mathbf{v} = \frac{1}{\lambda M} \sum_{i=1}^{M} \mathbf{x_i^T} \mathbf{v} \mathbf{x}_i$$

$$= \frac{1}{\lambda M} \sum_{i=1}^{M} \beta_i \mathbf{x}_i = \sum_{i=1}^{M} \alpha_i \mathbf{x}_i$$

Or in the simplest term, eigen vector (EV) is a linear combination of input samples.

Covariance Matrix in Feature Space In Kernel PCA, PCA is done in the feature space. For this, the data needs to be mapped into another space, i.e.,

$$\mathbf{x} \rightarrow \phi(\mathbf{x}).$$

Then assuming that data is centered, covariance matrix can be computed as:

$$C = \frac{1}{M} \sum_{j=1}^{M} \phi(\mathbf{x_j}) \phi(\mathbf{x_j})^T \tag{16.34}$$

16.101.2 EV as Linear Combination of Samples

For KPCA, we need to solve:

$$C\mathbf{V} = \lambda\mathbf{V} \tag{16.35}$$

- We do not want to work with $\phi(\cdot)$ explicitly. We can use the kernel $\kappa(\cdot,\cdot)$ and curcumvent this requriement.
- If $\phi(\cdot)$ maps to an infinite dimension, we will have to work with $\infty \times \infty$ matrices. That is impractical.
- Note that even in such cases, we will have ONLY $\leq M$ eigen vectors for C.

We can rewrite it as

$$\mathbf{V} = \frac{1}{\lambda}C\mathbf{V} = \frac{1}{\lambda M}\sum_{i=1}^{M}\phi(\mathbf{x_i})\phi(\mathbf{x_i})^{\mathbf{T}}\mathbf{V}$$

$$= \frac{1}{M}\sum_{i=1}^{M}(\phi(\mathbf{x_i})^{\mathbf{T}}\mathbf{V})\phi(\mathbf{x_i}) = \sum_i \alpha_i\phi(\mathbf{x_i})$$

Or

$$\mathbf{V} = \sum_{i=1}^{M}\alpha_i\phi(\mathbf{x_i})$$

Note that the upper case $\mathbf{V}$ is the eigen vector in the new feature space.

We do not want to compute the covariance matrix in the new feature space. We also do not want to compute this eigen vactor. We want only the projection of new samples into the new sub-spaces defined by the eigen vectors of the new covariance space.

16.101.3 Computing α

Taking a dot product with $\phi(\mathbf{x_k})$ on both sides of Equation 16.35: $(C\mathbf{V} = \lambda\mathbf{V})$

$$\phi(\mathbf{x_k}) \cdot C\mathbf{V} = \lambda\phi(\mathbf{x_k}) \cdot \mathbf{V} \;\; \forall k \tag{16.36}$$

Substituting for C, $\mathbf{V}$ and rearranging the summations,

$$\sum_{j=1}^{M}(\phi(\mathbf{x_k}) \cdot \phi(\mathbf{x_j}))\sum_{i=1}^{M}\alpha_i(\phi(\mathbf{x_j}) \cdot \phi(\mathbf{x_i})) \tag{16.37}$$

$$= \lambda M\sum_{i=1}^{M}\alpha_i(\phi(\mathbf{x_i}) \cdot \phi(\mathbf{x_k}))\forall k \tag{16.38}$$

Note we can do it for all $\mathbf{x}_k$, leading to M similar equations. We can compactly represent these equations in matrix form with kernel matrix K (our familiar one).

Defining an $M \times M$ matrix K by

$$K_{ij} = (\phi(\mathbf{x_i}) \cdot \phi(\mathbf{x_i})) = \kappa(\mathbf{x_i}, \mathbf{x_j})$$

We can compactly write these equations as:

$$M\lambda K\alpha = K^2\alpha$$
$$M\lambda\alpha = K\alpha$$

Thus the vectors α are the eigen vectors of K. Here, α is a vector of dimension M. K is a matrix of size $M \times M$, and it has M eigen vectors α. Let α^k be the eigen vector corresponding to λ_k.

The M eigen vectors of K, α^k lead to M eigen vectors of C.

The resulting set of eigenvectors $\mathbf{V}^k$ can now be computed as:

$$\mathbf{V^k} = \sum_{i=1}^{M}\alpha_i^k\phi(\mathbf{x_i}).$$

16.101.4 Projecting a New Sample

And the projection of a sample $\phi(\mathbf{x})$ onto this principal component can be evaluated as:

$$(\mathbf{V^k} \cdot \phi(\mathbf{x})) = \sum_{i=1}^{M}\alpha_i^k(\phi(\mathbf{x_i}) \cdot \phi(\mathbf{x}))$$
$$= \sum_{i=1}^{M}\alpha_i^k\kappa(\mathbf{x_i}, \mathbf{x})$$

16.101.5 Centering

We started with the assumption that the data is centered. In practice it is not. How do we center the data then?

Let $\hat{\phi}(\mathbf{x_i})$ be the centered version of $\phi(\mathbf{x_i})$. The (i,j) th element of the Kernel matrix (corresponding to centered data) is:

$$\hat{K}_{ij} = \hat{\phi}(\mathbf{x_i})^T\hat{\phi}(\mathbf{x_j}) \tag{16.39}$$

$$\hat{K}_{ij} = (\phi(\mathbf{x_i}) - \mu)^T(\phi(\mathbf{x_j}) - \mu)$$

Where $\mu = \frac{1}{N}\sum_{n=1}^{N}\phi(\mathbf{x_n})$

$$\begin{aligned}\hat{K}_{ij} &= (\phi(\mathbf{x_i}) - \mu)^T(\phi(\mathbf{x_j}) - \mu) \\ &= (\phi(\mathbf{x_i}) - \frac{1}{N}\sum_{m=1}^{N}\phi(\mathbf{x_m}))^T(\phi(\mathbf{x_j}) - \frac{1}{N}\sum_{n=1}^{N}\phi(\mathbf{x_n})) \\ &= K(\mathbf{x_i}, \mathbf{x_j}) - \frac{1}{N}\sum_{n=1}^{N}K(\mathbf{x_n}, \mathbf{x_i}) - \frac{1}{N}\sum_{m=1}^{N}K(\mathbf{x_m}, \mathbf{x_j}) \\ &+ \frac{1}{N^2}\sum_{m,n=1}K(\mathbf{x_m}, \mathbf{x_n})\end{aligned}$$

$$\hat{K}_{ij} = K_{ij} - \frac{1}{N}\sum_{n=1}^{N}K_{ni} - \frac{1}{N}\sum_{m=1}^{N}K_{mj} + \frac{1}{N^2}\sum_{mn}K_{mn}$$

16.101.6 Computational Procedure

1. First compute the kernel matrix K.
2. Then compute the kernel matrix $\hat{K}$ corresponding to the centered data.
3. Compute the eigen values and eigen vectors (α) of $\hat{K}$
4. Use α and the kernel function, evaluate the projection.

16.101.6.1 Kernel LDA and Beyond (*)

The kernel trick is not limited to PCA. You can also kernelaize algorithms like LDA. Please see:

- B. Scholkof, A Smola and K Muller, Nonlinear Component Analysis as a Kernel Eigenvalue Problem for KPCA
- S. Mika et al, Fisher Discriminant Analysis with Kernels for KLDA

16.102 Application: Back to Eigen Face

We had discussed PCA being a useful cue for representing faces in the form of eigen faces. The dimensionality reduction techniques like LDA (or Fisher discriminant) and KPCA are also used in very similar manner.

Q: Write pseudo codes for fisher and KPCA faces.

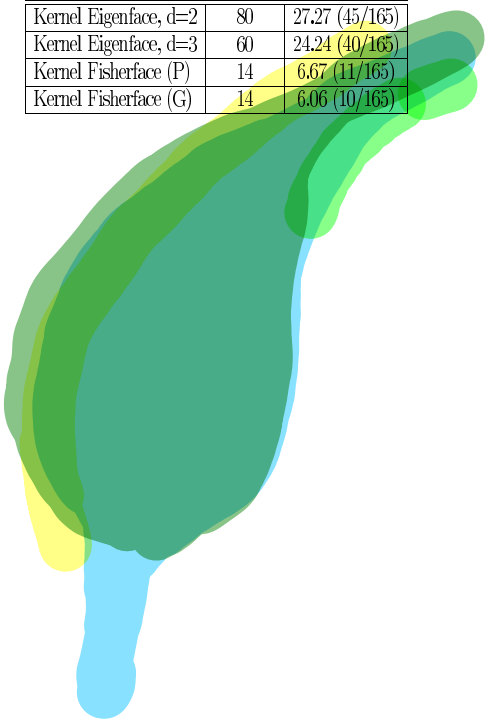
"Face recognition using kernel Methods" (NIPS 2001) is provided a nice comparison of these methods on two benchmarks. (see more details)

x

Method	Reduced Space	Error Rate (%)
ICA	40	6.25 (25/400)
Eigenface	30	2.75 (11/400)
Fisherface	14	1.50 (6/400)
Kernel Eigenface, d=2	50	2.50 (10/400)
Kernel Eigenface, d=3	50	2.00 (8/400)
Kernel Fisherface (P)	14	1.25 (5/400)
Kernel Fisherface (G)	14	1.25 (5/400)

Method	Reduced Space	Error Rate (%)
ICA	30	29.09 (48/165)
Eigenface	30	28.48 (47/165)
Fisherface	14	8.48 (14/165)
Kernel Eigenface, d=2	80	27.27 (45/165)
Kernel Eigenface, d=3	60	24.24 (40/165)
Kernel Fisherface (P)	14	6.67 (11/165)
Kernel Fisherface (G)	14	6.06 (10/165)

x



CSE 475: Statistical Methods in AI **Monsoon 2019**

SMAI-M-2019 17: Neural Networks: Multi Layer Perceptrons

Lecturer: C. V. Jawahar *Date: 10 Oct, 2019*

17.103 Introduction

A neural network (or more precisely artificial neural network (ANN)) consists of a set of neurons arranged in a specific manner. Based on the mathematical model of the neuron, their organisation and the learning algorithm employed, the network becomes capable of solving various pattern recognition tasks. A neuron typically takes inputs from various neurons and outputs its activation state. Inputs are usually weighed and added within a neuron. Let w_{kj} is the weight of connection between jth and kth neurons. Adder within the neuron sums the weighted inputs. Output is a nonlinear activation function of this weighted sum. Activation function also limits the value of the output signal to some finite value

Neural netwrok as a computational model has many contrasting difference with the traditional ones including Sequential Vs. Parallel. CPUs are complex architectures while neurons are simple structures. Neurons are typically 5 to 6 order slower than silicon gates. While Number of instructions per sec. is a popular measure of complexity in the traditional computing, number of interconnections or number of weights has very strong role in the neural netwroks. Traditionally, we formulate a mathematical model and validate with real data. In NN learning, we identify the model directly from the data

17.103.1 Biological Neural Networks

Neural networks are inspired by our brains. The human brain has about 10^{11} neurons and 10^{14} synapses. A neuron consists of a soma (cell body), axons (sends signals), and dendrites (receives signals). A synapse connects an axon to a dendrite. Given a signal, a synapse might increase (excite) or decrease (inhibit) electrical potential. Neurons are densely interconnected with each other. Connections between neurons need not be simple and layered. Each neuron could be receiving inputs from approximately 10^5 synapses.

A neuron receives signals from connected neurons via dendrites. The cell body sums up the received signals. The neuron then fires if the combined weighted input exceeds a threshold. By neuron firing, we mean that it generates an output which is sent out through axons to the next set of neurons. If the weighted sum is below the threshold, no response signal is generated by the neuron (i.e., the output is zero or neuron did not fire). The threshold decides whether a neuron fires or not is called activation function. In other words, a neuron fires when its electrical potential reaches a threshold. Learning might occur by changes to synapses.

17.103.2 Brief History of ANNs

Neural network research has rich history. There are biological and cognitive motivation for understanding the neural netwroks. There are also engineering interest in understanding ANNs as a mathematical tool to model the complex systems. Neural networks have also gone through many ups and downs. They played big role in the success of deep learning and modern AI.

17.104 Artificial Neural Networks

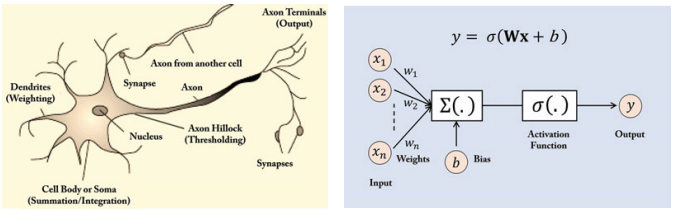


Figure 17.14: Biological and Arficial Neuron Models

17.104.1 Neuron Model

The computational model used popularly today is a simple weighted addition of inputs passing through a nonlinear function. i.e.,

$$y = \phi(\sum_{i=1}^{d} w_i x^i)$$

17.105 Feed Forward and Feedback Networks

Neural networks can be broadly classified as feed forward and feedback. In the case of feedforward networks, the signal flow (or information flow) takes place only in one direction. i.e., forward. If we look at this network as a graph, they are directed acyclic graphs. Examples include MLP and CNN. While in the case of feedback networks, output signal is fedback to the network, or sometime the same neuron. Examples include RNNs with popular implementations like LSTM and GRU. Due to the feedback nature of the connections, it some time becomes nontrivial to analyze, model or even understand such networks.

17.106 Multi Layer Perceptrons

17.106.1 Single Layer Perceptrons

We know about the linear classifiers that classify based on the simple rule:

$$sign(\mathbf{w}^T\mathbf{x}).$$

Such a classifier classifies a sample into positive class if $\mathbf{w}^T\mathbf{x}$ is ≥ 0 and negative class if $\mathbf{w}^T\mathbf{x}$ negative. We also know the perception algorithm that learns the $\mathbf{w}$ for a linear separable problem.

We can look at this as a single layer neural network with inputs as $x^1, x^2, \ldots, x^d$ and the weights (on the edges) $w_1, w_2, \ldots w_d$ gets multiplied and added.

There are two computations that are happening with in the neuron. (1) Multiply and add (i.e., compute $\mathbf{w}^T\mathbf{x}$. (2) pass through the nonlinearity $\phi()$, also known as the activation. In this case the nonlinearity $\phi()$ is

$$\phi(x) = \begin{cases} +1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

In this case $\phi(\cdot)$ is practically the $sign()$ function.

Single Layer Perceptron or simple perceptron has only one layer and the above mentioned activation function. This can classify samples into two class with a linear decision boundary.

Activation function: This nonlinearity $\phi()$ is often called activation function. A disadvantage of the above activation function is that it is not differentiable. A logistic/sigmoid function is often used as the nonlinearity

$$\phi(x) = \frac{1}{1+e^{-x}}$$

As shown in the figure 17.15, this function ranges between 0 and 1.

Figure 17.15: Sigmoid or Logistic Function $\phi(x) = \frac{1}{1+e^{-x}}$

17.106.2 MLP

In Multi Layer Perceptron (MLP), we connect or concatenate multiple perceptrons and model a complex function class. We can connect many single layer neural networks to form a multi layer neural network. A typical MLP has

- **Nodes and Edges** Network has nodes (where simple computations take place and edges where information flow happen.
- **Layered architecture.** Network is understood and represented in layers.
- **Dense Connections** Successive layers are often fully connected.

One way to appreciate a MLP/Neural network as a learniable function from X to Y. Eventually this network models the transformation of the input $\mathbf{x}_i$ to $\mathbf{y}_i$. From this point of view, MLPs can be used for either regression or classification. In other words, $\mathbf{y}$ could be a single intiger, real number or multiple integers/real numbers.

17.106.3 MLP for Classification

MLPs are very good for the classification. In the case of binary classification, one can design a network with only one output neuron. This neuron could output the probability being class-1. And a complement of this could be class-2. This can be achieved by keeping a sigmoid or tanh at the output neuron.

What about multiclass classification? Assume there are four classes, how many neurons are required in the output space? One solution could be to use two neurons and encode the classes as 00, 01, 10 and 11. However, this

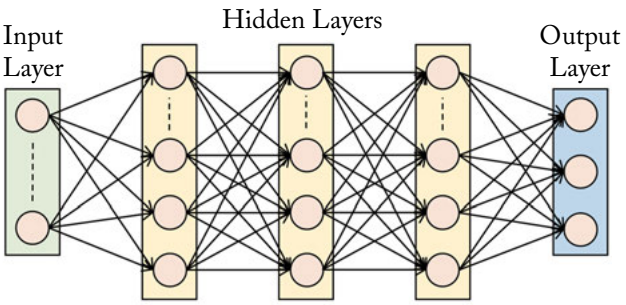


Figure 17.16: A typical MLP with three hidden layers.

assumes a structure at the output space (eg. a specific class is larger than the other). This is not preferred. The preferred solution is to have four output neuron and each outputting the probability (or confidence) for certain class. What if two classes are fired at a time.? To avoid this, one can use softmax at the output layer.

In other words, in the case of classification, it is the practice to represent the class-ID as a one hot vector. i.e., if there are C classes, there will be C neurons in the output. The the desired class is p, then we represent the output as a C dimensional vector with zero every where except at pth location. Output is often a softmax

$$\frac{e^{x^j}}{\sum_{i=1}^{K} e^{x^i}}$$

17.106.4 MLP for Regression

MLPs are popular for regression as well as classification. In the case of regression, the output neurons predict a real quantify.

In some cases, the output range need not be $[0, 1]$ or $[-1, 1]$. For example, we want to design a network that will predict the age of a person from a photograph. In such cases, output could be even a simple linear activation i.e., $x = \phi(x)$.

17.107 Notations

The layer to layer transition in a typical MLP can be understood as a matrix multiplication followed by the activation computation.

Let $\mathbf{x}$ is the input. Then the output of the next layer is represented as:

$$\mathbf{y} = \phi(\mathbf{W}\mathbf{x})$$

Representing the network in this manner (than using, i,j,k notations makes the representations compact.

17.108 Design Choices

17.108.1 Why do we need a nonlinearity?

In typical MLPs, it is assumed that all Neurons in a layer gets connected to all the neurons in the next layer. i.e., the layer is fully connected. Say $\mathbf{x_i}$ is the representation (output of the neurons at the ith layer and $\mathbf{x_{i+1}}$ is the representation at the $i+1$ th layer, then we can compute

$$\mathbf{x_{i+1}} = \phi(\mathbf{W}\mathbf{x}_i)$$

If there was no nonlinearity, then multiple layer perceptron could have reduced to a single layer perceptron.

Example: If $\mathbf{x_2} = \mathbf{W}'\mathbf{x}_1$ and $\mathbf{x_3} = W''\mathbf{x}_2$, then we can in fact write $\mathbf{x_3} = \mathbf{W}'''\mathbf{x_1}$. For some $\mathbf{W}''' = \mathbf{W}' \cdot \mathbf{W}''$

17.108.2 Architecture

A typical MLP what is given to us is the input, output pairs $(\mathbf{x_i}, \mathbf{y_i})$ $i = 1, \ldots N$. Typically $\mathbf{x}$ and $\mathbf{y}$ are vectors. The number of neurons in the first/input layer is the dimensionality of $\mathbf{x}$. Number of neurons in the output layer is the dimensionality of $\mathbf{y}$. We have two things in our control (i) Number of hidden layers (ii) number of neurons in each hidden layer. Typically, we go for 2 or 3 hidden layers. With number of layers increasing, the network becomes deep and the learning problem becomes difficult due to issues like vanishing gradient.

The number of neurons in each layer is typically larger than the number of neurons that are required at the input or output layers. Note that typical neural networks are over parameterized. i.e., there are more neurons and weights than what is required for solving the problem. (Indeed it may be possible to prune and get a smaller network once the network is trained.)

17.108.3 Loss Function

The objective of the network is to minimze the loss functions (or objective functions) during the training. The choice of loss function may depend on the problem. However, there are a set of popular loss functions that works for most problems.

Let there are K output neurons. Let the true output/target be $\mathbf{t}$ and the prediction/output is $\mathbf{o}$.

1. **MSE** Mean Square Error (MSE) is a popular loss function

$$\mathcal{L}(\mathbf{o}, \mathbf{t}) = \frac{1}{2N} \sum_{i=1}^{K} (t_i - o_i)^2$$

2. **L1**

$$\mathcal{L}(\mathbf{o},\mathbf{t}) = \frac{1}{N}\sum_{i=1}^{K}|t_i - o_i|$$

3. **Cross Entropy**

$$\mathcal{L}(\mathbf{o},\mathbf{t}) = \sum_{i=1}^{K} t_i \log(o_i)$$

If $\mathbf{t}$ and $\mathbf{o}$ are probability distributions (desired and output), then the cross entropy loss is equivalent to the KL divergence between these two distributions.

$$D_{KL}(\mathbf{t}||\mathbf{o}) = \mathcal{L}(\mathbf{o},\mathbf{t}) - H(\mathbf{t})$$

$$D_{KL}(\mathbf{t}||\mathbf{o}) = \mathcal{L}(\mathbf{o},\mathbf{t}) + \sum_i t_i \log t_i$$

$$D_{KL}(\mathbf{t}||\mathbf{o}) = \sum_i t_i \log(o_i) + \sum_i t_i \log t_i = \sum_i t_i \log(\frac{o_i}{t_i})$$

(double check equations!!)

- Q: Show that KL Divergence is non-negative
- Q: Is KL Divergence symmetric?
- Q: What is "Bhattacharya distance"?

4. **Hinge Loss**

$$\mathcal{L}(\mathbf{o},\mathbf{t}) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}\max(0, 1 - \mathbf{w}^T\mathbf{x}_i t_i)^2$$

Understand the Hinge loss as:

$$\max(0, 1 - wT.x_i.t_i)$$

where target t is ± 1 and o is the raw output (not thresholded).

- What happens when $t \cdot o$ is negative? sample is misclassified and the error is high.
- When the $t \cdot o$ is larger than 1? Loss is zero
- When $t \cdot o \in (0, 1)$?

Read: Deep Learning using Linear Support Vector Machines

5. **Regularized Losses**

$$\mathcal{L}_R = \mathcal{L} + ||\mathbf{w}||_2^2$$
$$\mathcal{L}_R = \mathcal{L} + ||\mathbf{w}||_1$$

Like in other problems, we can (and should regularize the weights).

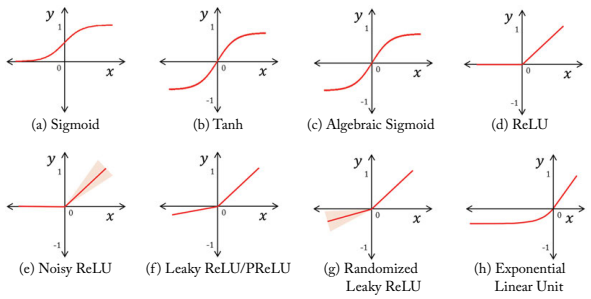


Figure 17.17: A set of popular activations/nonlinearities used in todays neural networks

17.108.4 Activation Functions

Here are some examples of popular activation functions:

1. **Sigmoid**

$$y = \frac{1}{1 + e^{-x}}$$

2. **tanh**

$$y = \frac{e^x + e^{(}-x)}{e^x + e^x}$$

$$y = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

3. **ReLu**

$$y = \max(0, x)$$

4. **Leakly Relu**

$$y = \begin{cases} x & \text{if } x > 0 \\ cx & \text{if } x \leq 0 \end{cases}$$

5. **Noisy ReLu**

$$y = \max(0, x + \epsilon)$$

$$\epsilon = \mathcal{N}(0, \sigma(x))$$

6. **Exponential Linear Unit**

$$y = \begin{cases} x & \text{if } x > 0 \\ a(e^x - 1) & \text{if } x \leq 0 \end{cases}$$

- Q: Write the expression for the other two activations.
- Q: Write expressions for $\phi'(x)$

17.109 Expressive Power of MLP

17.109.1 AND and OR

Consider the problem of implementing "AND" and "OR" with single layer perceptrons.

x^1	x^2	AND	OR	EXOR
0	0	0	0	0
0	1	0	1	1
1	0	1	0	1
1	1	1	1	0

Let us implement these logics with a single layer neural network with an activation $\phi(x) = 1$ iff $x \geq 0$. With the introduction of bias, what we want to lean is $w_0, w_1 and w_2$.

It can be seen that the following weights satisfy:

- **AND**: $w_0 = w_1 = w_2 =$
- **OR**: $w_0 = w_1 = w_2 =$

Q: Are they the only possible weights that satisfy?

Q: Can we get a valid solution with no bias?

Q: Can you find weights corresponding to NAND and NOR?

Q: Here, we used $0, 1$ logic. Assume we use $-1, 1$ representation, can you redesign the networks?

If we plot this data, we can see that these are linearly separable.

17.109.2 EXOR

However, that is not true for EXOR. It can be easily seen that no solution of the form $w_2x^2 + w_1x^1 + w_0$ is going to work for EXOR.

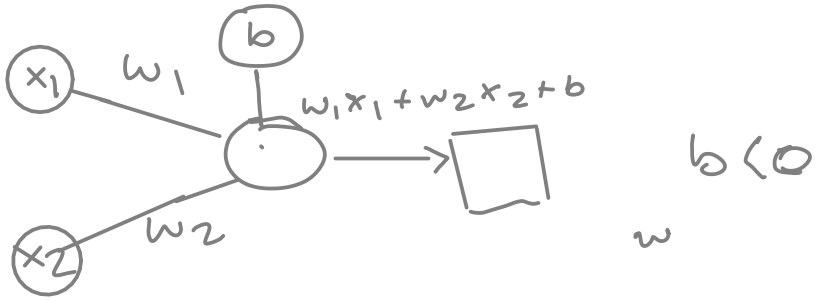
17.109.3 Representational Power

17.110 Learning

The key question is on "How do we train the neural network." This is possibly more important than how do we decide the number of neurons in each layer or number of layers for a beginner.

Even for an experiences, the learning process is not that simple. Experience in carefully doing an experiment is required to get the best.

The popular algorithm for training neural networks is called "Error Backpropagation Algorithm" or popularly known as "backpropagation algorithm (BP)".



Homeworks

1. Design a neural network each with two inputs and one output to implement AND, OR, ExOR, NAND and NOR logics with +1,-1 convention. Neurons have an activation of $sign()$. Write Truthtables, Draw architectures, write weights and any other associated details.

2. We know how to design a network that can solve ExOR problem. Let us consider an extension of it from 2 input to four, and cast the problem as "parity" problem. (i.e., whether the number of 1s are even or odd.). Write the truth table. Design a network with $sign()$ activations. (if required, write some code while designing!).

3. Consider a sequential data (like price of a vegitable over months). This vary over time and often is a sequence. However, there is some pattern in this sequence. Let us make a simple assumption and create a data as:

$$x(n) = \alpha_1 x(n-1) + \alpha_2 x(n-2) + \alpha_3 x(n-3) + \alpha_4 x(n-4) + \alpha_5 x(n-5) + N(0, \sigma^2)$$

Choose different values for α_i (including negative) and also appropriately taking care of initial conditions, plot this data (notice some trend, patterns in the data!)

Now we would like to model this problem as a regression problem that minimize

$$\mathcal{E}(\mathbf{w}) = \sum_k (x(k) - \sum_{i=1}^{d} w_i x(k-i))^2$$

And solve this problem using SLP.

(a) Find situations where we are able to reliably estimate the model (and of course do prediction of priced in the future) in presence of noise. Show true and one step ahead predicted data plotted toghether.

(b) Show some situations where this prediction fails. Why?

(c) Start with a data where reliable estimate of α is feasible. Now create a plot of $\mathcal{E}$ vs d. What do we observe here?

$w_1 = w_2$

0 0 $\text{sign}(b) = 0$

0 1 $\text{sign}(w_2 + b) = 0$

1 0 $\text{sign}(w_1 + b) = 0$

1 1 $\text{sign}(w_1 + w_2 + b) = 1$

CSE 475: Statistical Methods in AI **Monsoon 2019**

SMAI-M-2019 18: Backpropagation

Lecturer: C. V. Jawahar *Date: DATE*

18.111 High Level Picture

Let us consider an MLP as a sequence/chain of computational blocks. (see figure 18.18). In practice, these blocks have a matrix multiplication and an activation function of the form $\mathbf{x}_{n+1} = \phi(\mathbf{W_n}\mathbf{x}_n)$. Here, $\mathbf{x}_n$ corresponds to the number of neurons in the n th layer. Note that the number of neurons in each layer may be different. This implies that $\mathbf{W}_n$ matrix need not be square. Needless to say, they could be different for each layer.

There is a loss layer at the end of the chain. This module computes the discrepancy of the last output ($\mathbf{x}_{p+1}$) with the expected value ($\mathbf{y}$) and compute a scalar loss measure.

The objective of learning is to find the parameters (i.e., $\mathbf{W}$ matrices) that minimize the loss.

Assuming that there are p layers, we have matrices $\mathbf{W}_1, \ldots \mathbf{W}_p$ that parameterize the neural network. In otherwise, we have that many parameters to learn.

Our gradient descent learning rule will allow us to learn in the form of

$$\mathbf{W}^{k+1} \leftarrow \mathbf{W}^k - \eta \frac{\partial L}{\partial \mathbf{W}^k} \tag{18.40}$$

As in the previous gradient descent schemes, we can start with a random (or preferably a smart) initialization of the weight matrices in the zero iteration (initialization) and update it with every iteration k, until some convergence criteria is met.

However, the problem is not simple. The loss depends only on the $\mathbf{x}_{p+1}$ and the true prediction $\mathbf{y}$. Then how can the partial derivatives in equation 18.40 be nonzero?. On a closer look, we realize that $\mathbf{x}_{p+1}$ depends on the previous weight matrix $\mathbf{W}_p$, and also $\mathbf{x}_p$.

18.112 Derivatives

Computation of the partial derivatives is not that complex, if we use our familiar chain rule. We make an assumption at this stage:

1. **criteria -A** For each block, we know how to compute the partial derivative of the output with respect to that of input.

   $$i.e., \frac{\partial \mathbf{x}_{n+1}}{\partial \mathbf{x}_n}$$

2. **criteria -B** For each block we know how to compute the partial derivative of the output with respect to that of the learnable parameters (say $\mathbf{W}$).

   $$i.e., \frac{\partial \mathbf{x}_{n+1}}{\partial \mathbf{W_n}}$$

   Some of the blocks may also have learnable parameters θ other than weights. In such case, we also should know the

   $$i.e., \frac{\partial \mathbf{x}_{n+1}}{\partial \theta}$$

We can compute the partial derivative of the loss with respect to any of the learnable parameters using the chain rule. This allows us to learn the weights using the gradient update rule in equation 18.40.

18.112.1 Loss Layer

Consider the final loss layer which computes loss from $\mathbf{x}_{p+1}$ and $\mathbf{y}$ for each sample. An example of the loss is

$$\mathcal{L} = \sum_{i=1}^{N} ||\mathbf{x}_{p+1} - \mathbf{y}||^2 = \sum_{i=1}^{N} [\mathbf{x}_{p+1} - \mathbf{y}]^T [\mathbf{x}_{p+1} - \mathbf{y}] \tag{18.41}$$

In this case $\frac{\partial \mathcal{L}}{\mathbf{x_p}}$ is easily computable. Note that $\mathbf{y}$ is constant/fixed. It is part of the data or gound truth.

Q: Do compute $\frac{\partial L}{\mathbf{x_p}}$ for three popular loss functions. (or loss functions that you think are meaningful.)

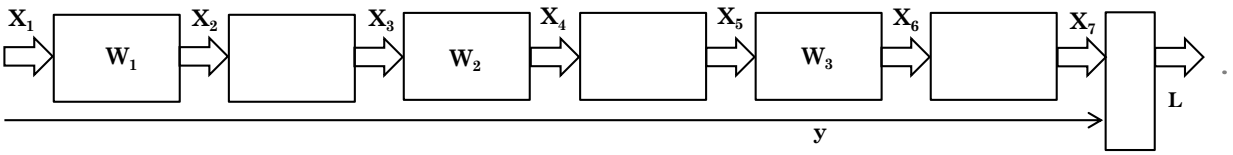


Figure 18.18: MLP as a sequence of computational blocks. Input $\mathbf{x}$ is same as $\mathbf{x_1}$. Output $\mathbf{x}_{p+1}$ is compared to the true output in the loss layer.

18.112.2 Typical Fully Connected Layer

A typical fully connected later (i.e., all neurons in layer k is connected to all the neurons in layer $k+1$) can be represented as

$$\mathbf{x}_{n+1} = \phi(\mathbf{W_n}\mathbf{x}_n)$$

To make the equations simpler, let us define a temporary variable $\mathbf{t}$, and rewrite the above as two steps.

$$\mathbf{t} = \mathbf{W_n}\mathbf{x}_n \tag{18.42}$$

$$\mathbf{x}_{n+1} = \phi(\mathbf{t}) \tag{18.43}$$

We need to compute $\frac{\partial \mathbf{x}_{n+1}}{\mathbf{W_n}}$ and $\frac{\partial \mathbf{x}_{n+1}}{\mathbf{x_n}}$.
These two are nothing but:

$$\frac{\partial \mathbf{x}_{n+1}}{\partial \mathbf{W_n}} = \phi'(\mathbf{t}) \cdot \frac{\partial \mathbf{t}}{\partial \mathbf{W_n}} \tag{18.44}$$

$$\frac{\partial \mathbf{x}_{n+1}}{\partial \mathbf{x_n}} = \phi'(\mathbf{t}) \cdot \frac{\partial \mathbf{t}}{\partial \mathbf{x_n}} \tag{18.45}$$

All the partial derivates on the right side are straightforward to compute.

$\phi'(\mathbf{t})$ is a vector of $\phi'(t_i)$. i.e., element-wise evaluation of the derivative.

Note: (i) Look some where else for how the matrix vector derivates are computed. Tom Minka's notes shared in the past is worth. (ii) See discussions somewhere else for How to compute $\phi'()$ for some of the popular activation functions.

18.113 Forward and Backward Passes

We are given $(\mathbf{x_i}, \mathbf{y_i})$ $i = 1, \ldots, N$. Our objective is to learn the weight matrices $\mathbf{w}_i$.

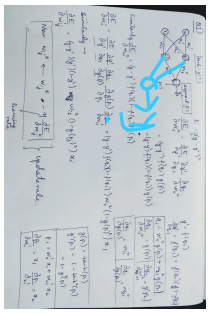
18.113.1 Forward Pass

For each sample in the training data we can give $\mathbf{x}_i$ as input and go through it through a series of matrix multiplications and activations. Finally the network predicts $\mathbf{x}_{p+1}$. We compute the loss per sample using equation 18.41 or similar other loss equations. Finally the total loss $\mathcal{L}$ is computed as the sum of loss over all the samples.

Forward pass is straightforward. It involves many matrix multiplications. This leaves scope for parallelization and running on dedicated hardware at high speed.

18.113.2 Backward Pass

Example 1 Let us consider the situation, we want to update the weight matrix of the last block as



$$\mathbf{w}_p^{k+1} \leftarrow \mathbf{W}_p^k - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}_p}$$

How do we compute $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_p}$?

Chain rule helps us to compute $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_p}$ as

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_p} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_p} \cdot \frac{\partial \mathbf{x}_p}{\partial \mathbf{W}_p}$$

The first term is available (see the text next to equation 18.41) and the next term is available with the definition of the block (see criteria B).

Example 2 Now let us try updating the weights in the last but one block.

$$\mathbf{w}_{p-1}^{k+1} \leftarrow \mathbf{W}_{p-1}^k - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}_{p-1}}$$

Similar to the previous case, we can compute

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{p-1}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_p} \cdot \frac{\partial \mathbf{x}_p}{\partial \mathbf{x}_{p-1}} \cdot \frac{\partial \mathbf{x}_{p-1}}{\partial \mathbf{W}_{p-1}}$$

We already know the availability of the first and last term (from the previous example of updating $\mathbf{W}_p$). We also know that the middle term is available from our criteria-A.

Example 3 Now we can compute

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_1} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_p} \cdot \frac{\partial \mathbf{x}_p}{\partial \mathbf{x}_{p-1}} \cdot \ldots \frac{\partial \mathbf{x}_3}{\partial \mathbf{x}_2} \cdot \frac{\partial \mathbf{x}_2}{\partial \mathbf{W}_1}$$

The point to note in the backward computation is that the partial derivatives required for the computation is available already, if we have updated the weights backwards.

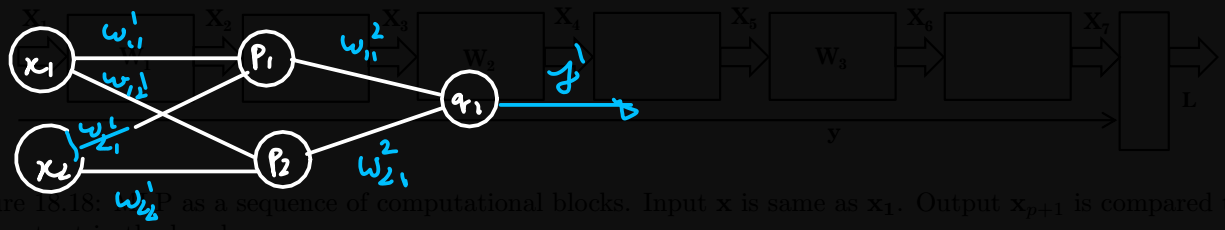


Figure 18.18: MLP as a sequence of computational blocks. Input $\mathbf{x}$ is same as $\mathbf{x_1}$. Output $\mathbf{x}_{p+1}$ is compared to the true output in the loss layer.

18.112.2 Typical Fully Connected Layer

A typical fully connected layer (i.e., all neurons in layer k is connected to all the neurons in layer $k+1$) can be represented as

$$\mathbf{x}_{n+1} = \phi(\mathbf{W_n}\mathbf{x}_n)$$

To make the equations simpler, let us define a temporary variable $\mathbf{t}$, and rewrite the above as two steps.

$$\mathbf{t} = \mathbf{W_n}\mathbf{x}_n \tag{18.42}$$

$$\mathbf{x}_{n+1} = \phi(\mathbf{t}) \tag{18.43}$$

We need to compute $\frac{\partial \mathbf{x}_{n+1}}{\mathbf{W_n}}$ and $\frac{\partial \mathbf{x}_{n+1}}{\mathbf{x_n}}$. These two are nothing but:

$$\frac{\partial \mathbf{x}_{n+1}}{\partial \mathbf{W_n}} = \phi'(\mathbf{t}) \cdot \frac{\partial \mathbf{t}}{\partial \mathbf{W_n}} \tag{18.44}$$

$$\frac{\partial \mathbf{x}_{n+1}}{\partial \mathbf{x_n}} = \phi'(\mathbf{t}) \cdot \frac{\partial \mathbf{t}}{\partial \mathbf{x_n}} \tag{18.45}$$

All the partial derivates on the right side are straightforward to compute.

$\phi'(\mathbf{t})$ is a vector of $\phi'(t_i)$. i.e., element-wise evaluation of the derivative.

Note: (i) Look some where else for how the matrix vector derivates are computed. Tom Minka's notes shared in the past is worth. (ii) See discussions somewhere else for How to compute $\phi'()$ for some of the popular activation functions.

18.113 Forward and Backward Passes

We are given $(\mathbf{x_i}, \mathbf{y_i})$ $i = 1, \ldots, N$. Our objective is to learn the weight matrices $\mathbf{w}_i$.

18.113.1 Forward Pass

For each sample in the training data we can give $\mathbf{x}_i$ as input and go through it through a series of matrix multiplications and activations. Finally the network predicts $\mathbf{x}_{p+1}$. We compute the loss per sample using equation 18.41 or similar other loss equations. Finally the total loss $\mathcal{L}$ is computed as the sum of loss over all the samples.

For a MLP, forward pass is straightforward. It involves many matrix multiplications. This leaves scope for parallelization and running on dedicated hardware at high speed.

18.113.2 Backward Pass

Example 1 Let us consider the situation, we want to update the weight matrix of the last block as

$$\mathbf{w}_p^{k+1} \leftarrow \mathbf{W}_p^k - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}_p}$$

How do we compute $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_p}$?

Chain rule helps us to compute $\frac{\partial \mathcal{L}}{\partial \mathbf{W}_p}$ as

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_p} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_p} \cdot \frac{\partial \mathbf{x}_p}{\partial \mathbf{W}_p}$$

The first term is available (see the text next to equation 18.41) and the next term is available with the definition of the block (see criteria B).

Example 2 Now let us try updating the weights in the last but one block.

$$\mathbf{w}_{p-1}^{k+1} \leftarrow \mathbf{W}_{p-1}^k - \eta \frac{\partial \mathcal{L}}{\partial \mathbf{W}_{p-1}}$$

Similar to the previous case, we can compute

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_{p-1}} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_p} \cdot \frac{\partial \mathbf{x}_p}{\partial \mathbf{x}_{p-1}} \cdot \frac{\partial \mathbf{x}_{p-1}}{\partial \mathbf{W}_{p-1}}$$

We already know the availability of the first and last term (from the previous example of updating $\mathbf{W}_p$). We also know that the middle term is available from our criteria-A.

Example 3 Now we can compute

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}_1} = \frac{\partial \mathcal{L}}{\partial \mathbf{x}_p} \cdot \frac{\partial \mathbf{x}_p}{\partial \mathbf{x}_{p-1}} \cdot \ldots \frac{\partial \mathbf{x}_3}{\partial \mathbf{x}_2} \cdot \frac{\partial \mathbf{x}_2}{\partial \mathbf{W}_1}$$

The point to note in the backward computation is that the partial derivatives required for the computation is available already, if we have updated the weights backwards.

18.114 Backpropagation

The error backpropagation or backpropagation can be summarized as:

1. Initialize the network with random weights.
2. For all the samples, compute the output of the neural network $\mathbf{x}_{p+1}$
3. Compute the loss for the full batch of N samples as

$$\mathcal{L} = \frac{1}{N}\sum_{i=1}^{N} L(\mathbf{x}_{p+1}^i, \mathbf{y}_i)$$

4. Adjust all the parameters (such as weight matrices) as

$$\theta^{k+1} \leftarrow \theta^k - \Delta\theta$$

or

$$\theta^{k+1} \leftarrow \theta^k - \eta\frac{\partial L}{\partial \theta}$$

5. Repeat steps 2-4 until convergence.

18.114.1 Refinements over backpropagation algorithm

Over years, backpropagation algorithm has been refined significantly with many minor but critical innovations. What all can change in the simple version we saw early?

1. **step 1:** Initialization can be smarter.
2. **step 2:** Computing loss over a full batch and updating it once is not the best.
3. **step 3:** Loss function can be different. There re many other loss functions available beyond MSE.
4. **step 4:** Update rule can be different. What we saw here is too simple.

18.115 Closer Look at the Derivatives (*)

Now that we had seen the larger picture of backpropagation and the chain rule for computing the derivatives, let us have a closer look at the equations 18.44 and 18.45.

$$t_n = W_n\mathbf{x}_n \quad ; \quad \mathbf{x}_n = \phi(\mathbf{t}_n)$$

$$\mathbf{x}_{n+1} : q \times 1 \quad ; \quad \mathbf{x}_n : p \times 1 \quad ; \quad W_n : q \times p \quad ; \quad \mathbf{t}_n : q \times 1$$

$$\frac{\partial \mathbf{x}_{n+1}}{\partial \mathbf{x}_n} = \frac{\partial \mathbf{x}_{n+1}}{\partial \mathbf{t}_n}\frac{\partial \mathbf{t}_n}{\partial \mathbf{x}_n} \tag{18.46}$$

$$\frac{\partial \mathbf{x}_{n+1}}{\partial W_n} = \frac{\partial \mathbf{x}_{n+1}}{\partial \mathbf{t}_n}\frac{\partial \mathbf{t}_n}{\partial W_n} \tag{18.47}$$

It should be noted that in equation 18.47, even though $\mathbf{t}_n$ is of dimension $q \times 1$ and W_n is dimension $q \times p$, the derivative $\frac{\partial \mathbf{t}_n}{\partial W_n}$ has a maximum of $q \times p$ non-zero values. This is because, the i^{th} element of $\mathbf{t}_n$ depends only on the i^{th} row of W_n. As such, for the purpose of equation 18.47, we assume that the derivative $\frac{\partial \mathbf{t}_n}{\partial W_n}$ is represented by a $q \times p$ matrix with the i^{th} row containing the derivative of the i^{th} element of $\mathbf{t}_n$ with respect to the i^{th} row of W_n. Also to be noted here is that the derivative $\frac{\partial \mathbf{x}_{n+1}}{\partial \mathbf{t}_n}$ would be a $q \times q$ dimensional diagonal matrix, because the i^{th} element of $\mathbf{x}_{n+1}$ depends only on the i^{th} element of $\mathbf{t}_n$.

18.116 Training Process

18.117 More on Backpropagation

We had seen the back-propagation algorithm as one that iteratively minimizing the loss/error over the samples. We discussed the algorithm as two steps:

1. **Forward Pass:** Do a forward pass for all the samples and compute the loss over the training set.
2. **Backward Pass:** Do refine all the learnable parameters (weights) iteratively as:

$$\theta^{k+1} \leftarrow \theta^k - \eta\frac{\partial J}{\partial \theta}$$

To make the discussions and notations simple, we assume that all the learnable parameters are part of θ, and the loss/objective is $J(\theta)$. We use this notation throughout.

18.118 Stochasticity

In practice we do not compute the loss over all the samples and then update the parameters in one go. We do this over a randomly selected subset of the samples. This leads to stochastic mini batch backpropagation algorithm.

Note that the batch mode of the BP computes loss over all the samples. This is actually an approximation of the "true" gradient which we are not able to compute, since we do not know the analytic function form. If the true gradient can be approximated with sum of gradients over

a number of samples, thus approximation can be computed from a subset of the samples also. However, computing the gradient from a smaller set may have larger error than that computed from all the samples. However, this is much more efficient. Therefore, we can have BP implemented as batch, single sample and mini-batch. Minibatch is preferred.

The convergence and properties of stochastic gradient descent methods have been analyzed in both convex minimization and stochastic approximation. In many related areas, it has been shown that the stochastic gradient descent will converge to the batch (not stochastic) estimates in many practical situations.

18.118.1 Epochs and Iterations

In neural network literature, it is common to use the word epoch. In the neural network terminology, one epoch consists of one forward pass and one backward pass of all the training examples. However, as we discussed above, batch size (i.e., the number of training examples in one forward/backward pass) could be much smaller than the entire data. Though it is possible to randomly create the batch size every time, it becomes computationally efficient to create random batches once (in the beginning of the training) and use the same batches throughout. When the batch size is large, the memory requirement of the training could increase.

18.119 Sub-gradients

Another issue is the sub-gradient. many functions that we use in the modern deep neural networks are not truely differentiable. Eg. ReLU. How do we handle these?

Eg1: ReLu A popular activation function is ReLU

$$\phi(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases}$$

Eg2: Hinge Loss Hinge loss has been a key component in the success of SVMs.

$$max(0, 1 - t \cdot y)$$

Though in these two cases, the function is not continuous, we can compute the derivatives at each point, except the point of discontinuity. This is done with the help of "subgradients". In mathematics, the subgradient, generalizes the derivative to convex functions which are not necessarily differentiable.

18.120 Homeworks

1. Consider a simple neural network with two input two hidden and one output neurons. Hidden nodes have tanh activation and output has sigmoid activation.

   There are four learnable parameters (weights). Let the first four be represented as $w_{11}^1, w_{12}^1, w_{21}^1, w_{22}^1$ and the second set of two be w_{11}^2 and w_{12}^2. Loss is MSE loss.

   Derive the gradient update rule for each of these six weights as:

   $$w_{ij}^k \leftarrow w_{ij}^k + ...$$

2. Create a set of 100 each samples in 2D in two classes (assume multivariate Gaussian; Data is not linearly separable).

   Randomly sample 80% for training and 20% for testing.

   - implement the above learning process (i.e., in Q1). Does it give satisfactory results?
   - improve your implementation by adding bias. Does it give satisfactory results?
   - Try out a popular implementation (in your favorite library). How does this result compare with the above? Discuss.
   - Show learning graphs in all the above cases.

3. Use MLP to classify 1 vs 2 in MNIST data that you have been using. Input is $28 \times 28 = 784$. Two hidden layer of the size 1000 and final output of size 1.

   Discuss why or whether this is effective.

CSE 475: Statistical Methods in AI **Monsoon 2019**

SMAI-M-2019 19: More on Backpropagation

Lecturer: C. V. Jawahar *Date: DATE*

We had seen the back-propagation algorithm as one that iteratively minimizing the loss/error over the samples. We discussed the algorithm as two steps:

1. **Forward Pass:** Do a forward pass for all the samples and compute the loss over the training set.
2. **Backward Pass:** Do refine all the learnable parameters (weights) iteratively as:

$$\theta^{k+1} \leftarrow \theta^k - \eta \frac{\partial J}{\partial \theta}$$

To make the discussions and notations simple, we assume that all the learnable parameters are part of θ, and the loss/objective is $J(\theta)$. We use this notation throughout.

19.121 Stochasticity

In practice we do not compute the loss over all the samples and then update the parameters in one go. We do this over a randomly selected subset of the samples. This leads to stochastic mini batch backpropagation algorithm.

Note that the batch mode of the BP computes loss over all the samples. This is actually an approximation of the "true" gradient which we are not able to compute, since we do not know the analytic function form. If the true gradient can be approximated with sum of gradients over a number of samples, thus approximation can be computed from a subset of the samples also. However, computing the gradient from a smaller set may have larger error than that computed from all the samples. However, this is much more efficient. Therefore, we can have BP implemented as batch, single sample and mini-batch. Minibatch is preferred.

The convergence and properties of stochastic gradient descent methods have been analyzed in both convex minimization and stochastic approximation. In many related areas, it has been shown that the stochastic gradient descent will converge to the batch (not stochastic) estimates in many practical situations.

19.121.1 Epochs and Iterations

In neural network literature, it is common to use the word epoch. In the neural network terminology, one epoch consists of one forward pass and one backward pass of all the training examples. However, as we discussed above, batch size (i.e., the number of training examples in one forward/backward pass) could be much smaller than the entire data. Though it is possible to randomly create the batch size every time, it becomes computationally efficient to create random batches once (in the beginning of the training) and use the same batches throughout. When the batch size is large, the memory requirement of the training could increase.

19.122 Sub-gradients

Another issue is the sub-gradient. many functions that we use in the modern deep neural networks are not truely differentiable. Eg. ReLU. How do we handle these?

Eg1: ReLu A popular activation function is ReLU

$$\phi(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases}$$

Eg2: Hinge Loss Hinge loss has been a key component in the success of SVMs.

$$max(0, 1 - t \cdot y)$$

Though in these two cases, the function is not continuous, we can compute the derivatives at each point, except the point of discontinuity. This is done with the help of "subgradients". In mathematics, the subgradient, generalizes the derivative to convex functions which are not necessarily differentiable.

19.123 Refinements

19.123.1 Initialization

. Initialization is very important for any iterative solution to the non-convex optimization. It is always feared

that performance of the neural networks could be highly dependent on this lucky initialization. (Is it really true? Did you find so?)

Q: If we initialize all the weights as zero. What would happen? If we initialize all the weights as non-zero, but a constant value, what could happen?

It is observed that the random initializations are ideal for the neural networks. In the earlier days, it was common to try multiple initializations and pick the best. Even if we randomly initialize the weights, the output of a neuron depends on the inputs. The variance of the output directly depends on the inputs. Glorot and Bengio (2010) proposed to randomly initialize the weights with a variance that depends on the number of incoming (fan-in) and outgoing (fan-out) connections. This method (popularly known as Xavier's initilaization (AISTAT 2010) works well in practice.

19.123.2 Better Update Rules

We know the update rule as:

$$\theta^{k+1} \leftarrow \theta^k - \Delta\theta$$

Gradient Update In the simple gradient descent (and backpropagation) we saw, the change in weight $\Delta\theta$ is proportional to the derivative of the loss/objective.

$$\Delta\theta = \eta \frac{\partial L}{\partial \theta}$$

Momentum Momentum based optimization provided better convergence properties, and results. The idea is simple. If we conistently change the weights/parameters in certain direction, we can make larger steps instead of small steps.

$$\Delta\theta = \eta \frac{\partial L}{\partial \theta} + \gamma \Delta\theta^{t-1}$$

If we change the directions frequently, then we make small step. Connecting to the analogies from physics, we make big steps, if the surface is smooth. If surface is rough, we make small steps. Typically the momentum is set to 0.9. This classical momentum term could carry the ball beyond the minimum point. Ideally, we would like "the ball" to slow down when the ball reaches the minimum point and the slope starts increasing. This is achieved by the Nesterov Momentum.

Another aspect is that we have only one parameter for the learning rate. Can we have different parameters/scales for each dimension? Adaptive gradients and its variations are aimed at this.

Other Recent Advances(*) A number of refinements have surfaced in the recent years:

- Nesterov Momentum [Nesterov 1983]
- AdaGrad [Duchi 2011]
- AdaDelta [Zeiler 2012]
- RMSProp [Tieleman and Hinton, 2012]

Though some of these refinements were recent, they have all reached the end users through the popular libraries for neural networks.

The ADAptive Moment Estimation (ADAM) [Kingma and Ba, ICLR 2014] is a popular refinement of the update rule similar to the momentum techniuques. The main difference between Adam and its two predecessors (RMSprop and AdaDelta) is that the updates are estimated by using both the first moment and the second moment of the gradient. A running average of gradients (mean) is maintained along with a running average of the squared gradients.

19.123.3 Termination Criteria

Termination is often when there is no significant change in the loss/objective. However, a larger number of iterations can lead to overfitting. Often an "early stop" is performed by looking at how the loss/objective change over the validation data set. When the loss starts increasing on the validation data, iterations are stopped.

19.124 Loss Functions

We know loss function as a measure of discrepancy between the ground truth (true value) and the predicted output. The mean square error

$$\sum_{i=1}^{N} (t_i - o_i)^2$$

is a useful measure in this regard. This is popular for many regression tasks, where t_i and o_i are reals. If there are more than one neurons in the output layer (say M), then one can compute it by summing up over all the M neurons

$$\sum_{i=1}^{N} \sum_{j=1}^{M} (t_i^j - o_i^j)^2$$

This is not the apt measure for classification related tasks. When the output has M neuons and output of these neurons corresponds to the probabilities to these classes, then it is typical to use softmax in the output layer.

$$o^k = \frac{e^{o^k}}{\sum_{j=1}^{M} e^{o^j}}$$

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

Figure 19.19: Adam: Algorithm. taken from Kingma and Ba, ICLR 2014

. The softmax output o^k could be considered as the probability to be in class k. p^k.

The popular loss in this case is cross entropy loss

$$= \sum_{j=1}^{M} y^j \log p^j$$

for a M class classification problem. To appreciate this loss better, let us look at how can we compute the distance/dissimilarity between two probabilistic distributions. The classical measure for this is called Bhattacharya distance (or some people also call it as KL-divergemce)

$$D_{KL}(p(x)||q(x)) = \sum_x p(x) \ln \frac{p(x)}{q(x)}$$

A symmetric version $(D_{KL}(p(x)||q(x)) + D_{KL}(q(x)||p(x)))$ is also preferred in many cases. See Wikipedia on KL divergnce to appreciate the relationship between cross entropy and KL divergence.

19.124.1 Regularization

It is common to regularize the neural networks in different ways to prevent overfitting. One classical method is to look for simple/small neural network that can solve the problem of interest. The simplicity of such a neural network is measured with the number of non-zero weights (L0 norm of the weights) or sum of absolute value of the weights (L1 norm of the weights). Both L0 and L1 norms are hard to work with in most cases. In practice L2 norm is a good choice and the new loss then becomes old loss plus the L2 norm of the weights in the neural networks.

$$J' = J + ||\theta||_2^2$$

Please note that the new term is an addition. (remember $u+v$) and the new update rule will be almost the same as the old update rule plus an additional term corresponding to the L2 norm of the weights.

One can attempt to minimize $L0$ norm by pruning (removing or explicitly making it as zero) some of the tiny weights. Usually removal of some of the tiny weights need not change the network performance (output values) significantly. One of two iterations of the backpropagation

algorithm can recover any loss in performance due to this. However, having zero elements in the weight matrices will not save memory or computation if your inherent data structure is matrix/tensor. One needs to use appropriate sparse matrix computing techniques in this case.

19.125 Second Order Methods

The popular backpropagation algorithm that we studied till now uses only the first order derivatives. It takes linear steps along the negative gradient. Assume we have knowledge about the curvature/shape of the curve, then we could obtain better estimate of the solution in every step. Second order method uses Hessian (matrix of second order derivatives) in every step. (Recollect our discussions related to second order gradient descent methods elsewhere.)

$$\mathbf{H} = \begin{bmatrix} \frac{\partial^2 J}{\theta_1^2} & \frac{\partial^2 J}{\theta_1 \theta_2} & \dots & \frac{\partial^2 J}{\theta_1 \theta_N} \\ \dots & \dots & \dots & \dots \\ \frac{\partial^2 J}{\theta_N \theta_1} & \frac{\partial^2 J}{\theta_N \theta_2} & \dots & \frac{\partial^2 J}{\theta_N^2} \end{bmatrix}$$

Remembering the truncated Taylor series expansion,

$$J(\theta) \approx J(\theta_0) + (\theta - \theta_0)^T \nabla J(\theta_0) + (\theta - \theta_0)^T H (\theta - \theta_0)$$

We could take the derivative and equate to zero:

$$\nabla_\theta J(\theta) = \nabla_\theta J(\theta_0) + H(\theta - \theta_0) = \mathbf{0}$$

or

$$\theta = \theta_0 - \mathbf{H}^{-1} \nabla_\theta J(\theta_0)$$

If the function was quadratic, this step directly takes us to the minimum. (Q: why?). If not, this takes us to the minima of the quadratic approximation.

In general, we need to now compute g, compute H and then update as $\theta^{k+1} \leftarrow \theta^k - \mathbf{H}^{-1}\mathbf{g}$ Where $\mathbf{g} = \nabla_\theta J(\theta)$

The above mentioned second order method (alias Newtons method) requires, at every iteration, calculating and then inverting the Hessian. If the network has N parameters, then inverting the Hessian is $O(N^3)$. This renders Newtons method impractical for the modern deep neural networks.

When the Hessian has negative eigenvalues, then steps along the corresponding eigenvectors are gradient ascent steps. To counteract this, it is possible to regularize the Hessian, so that the updates become:

$$\theta \leftarrow \theta^k - (\mathbf{H} + \alpha I)^{-1} \nabla_\theta J(\theta_0)$$

As α becomes larger, this turns into first order gradient descent with learning rate $\frac{1}{\alpha}$.

19.126 Discussions (*)

19.126.1 Vanishing and Exploding Gradients

We had seen the derivation of the gradients using chain rule. If the gradients are small, then the product can "vanish" very fast. Similarly if the gradients are larger, then the products can explode very fast.

When cost function has "cliffs", where small changes in θ, drastically change the cost function. (This usually happens if parameters are repeatedly multiplied together, as in recurrent neural networks.) Similar to exploding gradients, repeated multiplication of a matrices/vectors can cause vanishing gradients.

These problems of vanishing and exploding problems have bothered for long time in numerically training deep neural networks.

19.126.2 More on Second Order Methods

To get around the problem of having to compute and invert the Hessian, quasi-Newton methods are often used. Amongst the most well-known is the BFGS (Broyden Fletcher Goldfarb Shanno) update.Quasi-Newton methods usually require a full batch (or very large mini-batches) since errors in estimating the inverse Hessian can result in poor steps.

CG methods are also beyond the scope of this class, but we bring it up here in case helpful to look into further. Again, ECE 236C is recommended if youd like to learn more about these techniques.

- CG methods find search directions that are conjugate with respect to the Hessian, i.e., that $g_k^T H g_{k1} = 0$.
- It turns out that these derivatives can be calculated iteratively through a recurrence relation.
- Implementations of Hessian-free CG methods have been demonstrated to converge well (e.g., Martens et al., ICML 2011).

19.126.3 Further Reading

This area has many advanced reading material. Interested students may read the original papers, tutorial notes online. They are beyond the scope of this course.

Homeworks

1. Consider the following initialization for a binary classification problem (i) All weights as zero (ii) All weights as one (unit value) (iii) All weights as random. Which may do well in practice? Give your

reasons. Demonstrate it with a simple experiment. Write your experiment design and methodology on paper, and submit the corresponding code.

2. Consider a 2 input, 3 hidden layers with neurons as 3, 5, 3 respectively and 1 output neuron. All three hiddden layers have sigmoid nonlinearity.

   - Derive the backpropagation algorithm in matrix form with clear definition of matrices and the derivatives, when the loss is
     - MSE loss
     - Hinge loss
   - Implement your matrix/vector/tensor equations in python and implement a backpropagation algorithm. Test it on a synthetic data of two multivariate Gaussians in 2D. Submit loss vs iterations plot for both cases.

3. We know the notion of regularization. Let us see how to implement that:

   - Let us assume that we regularize the MSE loss with L2 norm. The new loss is now MSE + L2 norm of the weights. Derive BP algorithm and validate on your synthetic data set. Show how weights change and how the "accuracies" change with regularization.

     (You can submit plots of accuracy vs iterations for both regularized and non-regularized cases)
   - Let us assume we want to do an L0 regularization. Let us do in the following way.
     - We decay and reduce the weights systematically. Each step, we also retrain so that overall performance is not compromised. Convert this into idea into a computational procedure and validate.
     - (Read about the following two terms "weight decay" and "pruning" in neural networks)

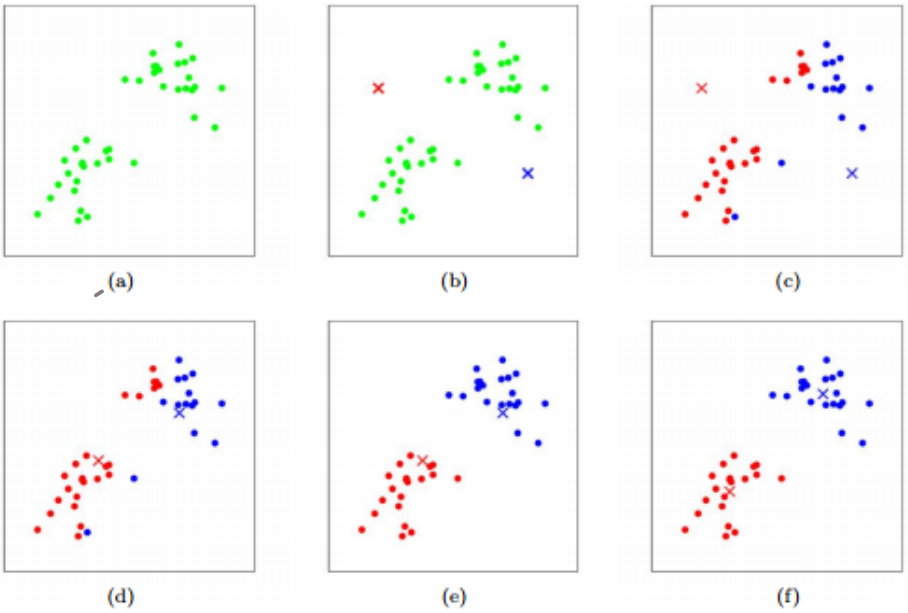
Unsupervised learning:

K-Means:

- Assume k classes
- Make k subsets of data
- Calculate mean
- relabel data according to distance.

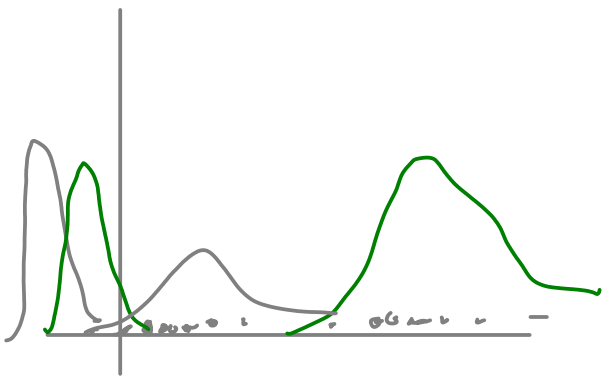
will converge

may not converge to global minimum.

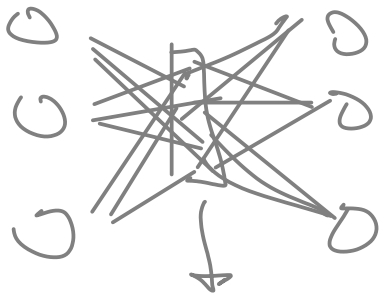


GMM:

$$r_{ik} = \frac{z_k N(x_i \mid \mu_k, \Sigma_k)}{\sum_{v=1}^{k} z_j N(x_i \mid \mu_v, \Sigma_v)}$$



Autoencoder & nonlinear PCA



representation

Ensembling

- multiple models
- majority voting

Boosting:

weak learners → strong learners

DTree

RF

many DTree