

¹ Lecture 2

² Linear Regression, ³ Perceptron, Stochastic ⁴ Gradient Descent

Reading

1. Bishop 3.1, 4.1, 4.3
2. Goodfellow Chapter 5.1-5.4
3. “Random features for large-scale kernel machines” by [Rahimi and Recht \(2008\)](#).

⁵ 2.1 Problem setup for machine learning

⁶ Nature gives us data X and targets Y for this data.

$$X \rightarrow Y.$$

⁷ Nature does not usually tell us what property of a datum $x \in X$ results in a
⁸ particular prediction $y \in Y$. We would like to learn to imitate Nature, namely
⁹ predict y given x .

¹⁰ What does such learning mean? It is simply a notion of being able to
¹¹ identify patterns in the input data without explicitly programming a computer
¹² for prediction. We are often happy with a learning process that identifies
¹³ correlations: if we learn correlations on a few samples $(x^1, y^1), \dots, (x^n, y^n)$,
¹⁴ we may be able to predict the output for a new datum x^{n+1} . We may not need
¹⁵ to know *why* the label of x^{n+1} was predicted to be so and so.

¹⁶ Let us say that Nature possesses a probability distribution P over (X, Y) .
¹⁷ We will formalize the problem of machine learning as Nature drawing n
¹⁸ independent and identically distributed samples from this distribution. This is

19 denoted by

$$D_{\text{train}} = \{(x^i, y^i) \sim P\}_{i=1}^n$$

20 is called the “training set”. We use this data to identify patterns that help make
21 predictions on some future data.

22 What is the task in machine learning?

23 Suppose D_{train} consists of $n = 50$ RGB images of size 100×100 of two kinds,
24 ones with an orange inside them and ones without. 10^4 is a large number of
25 pixels, each pixel taking any of the possible 255^3 values. Suppose we discover
26 that one particular pixel, say at location $(25, 45)$, takes distinct values in all
27 images inside our training set. We can then construct a predictor based on
28 this pixel. This predictor, it is a binary classifier, perfectly maps the training
29 images to their labels (orange: +1 or no orange: -1). If x_{ij}^k is the $(ij)^{\text{th}}$ pixel
30 for image x^k , then we use the function

$$f(x) = \begin{cases} y^k & \text{if } x_{ij}^k = x_{ij} \text{ for some } k = 1, \dots, n \\ -1 & \text{otherwise.} \end{cases}$$

31 This predictor certainly solves the task. It correctly works for all images in the
32 training set. Does it work for images outside the training set?

33 Our task in machine learning is to learn a predictor that works *outside* the
34 training set. The training set is only a source of information that Nature gives
35 us to find such a predictor.

Designing a predictor that is accurate on D_{train} is trivial. A hash function that memorizes the data is sufficient. This is NOT our task in machine learning. We want predictors that generalize to new data outside D_{train} .

36 2.1.1 Generalization

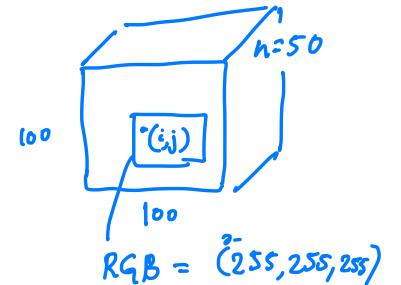
37 If we never see data from outside D_{train} why should we hope to do well on it?
38 The key is the distribution P . Machine learning is formalized as constructing
39 a predictor that works well on new data that is also drawn independently from
40 the distribution P . We will call this set of data the “test set”.

$$D_{\text{test}}.$$

41 This assumption is important. It provides coherence between past and future
42 samples: past samples that were used to train and future samples that we will
43 wish to predict upon.

44 How to find such predictors that work well on new data? The central idea
45 in machine learning is to restrict the set of possible binary functions that we
46 consider.

We are searching for a predictor that generalizes well but only have the training to ascertain which predictor works well.



❸ How many such binary classifiers are there at most?

47 The *right* class of functions f cannot be too large, otherwise we will find
 48 our binary classifier above as the solution and that is not too useful. The class
 49 of functions cannot be too small either, otherwise we won't be able to predict
 50 difficult images. If the predictor does not even work well on the training set,
 51 why should we expect it to work on the test set!

52 Finding this correct class of functions with the right balance is what
 53 machine learning is all about.

?

Can you now think how is machine learning different from other fields you might know such as statistics or optimization?

54 2.2 Linear regression

55 Let us focus on a simpler problem. We fix the class of functions, our predictors,
 56 to only have linear classifiers. We will consider that our data $X \subset \mathbb{R}^d$ and
 57 labels $Y \subset \mathbb{R}$. If the labels/targets are real-valued, we call it is a regression
 58 problem. Our predictor for any $x \in X$ is

$$f(x; w, b) = w^\top x + b. \quad (2.1)$$

59 This is a linear function in the data x with parameters $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$.
 60 Different settings of w and b give access to different functions f . Picking
 61 a particular function f is therefore akin to picking particular values of the
 62 parameters. Parameters are also called weights. We can visualize what this
 63 predictor does in two ways, consider the case of $d = 2$.

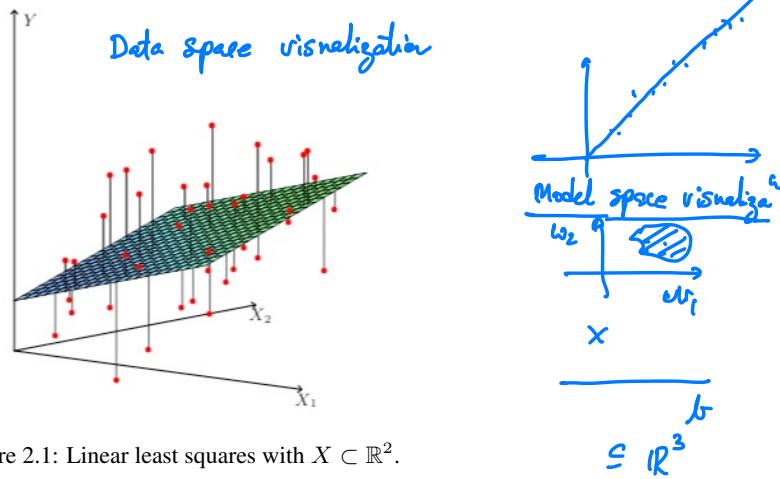
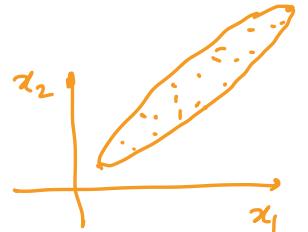


Figure 2.1: Linear least squares with $X \subset \mathbb{R}^2$.

64 Figure 2.1 shows the hyperplane corresponding to a particular (w, b) with
 65 the data x^i, y^i (in red). Each hyperplane is a particular predictor $f(x; w, b)$.
 66 You can also think of the function f as a point in three dimensional space
 67 $w \in \mathbb{R}^2$ and $b \in \mathbb{R}$.

68 Predicting the target accurately using this linear model would require us
 69 to find values (w, b) that minimize the average distance to the hyperplane of
 70 each sample in the training dataset. We write this as an *objective function*.

$$\begin{aligned} \ell(w, b) &:= \frac{1}{2n} \sum_{i=1}^n \left(y^i - \hat{y}^i \right)^2 \\ &= \frac{1}{2n} \sum_{i=1}^n \left(y^i - \underbrace{w^\top x^i - b}_{\text{Residuals}} \right)^2 \end{aligned} \quad (2.2)$$

71 where we have written the prediction as

$$\hat{y}^i = x^\top x^i + b.$$

72 The quadratic term for each datum $\frac{1}{2} (y^i - \hat{y}^i)^2$ is known as the *loss function*.

73 The objective above is thus an average of the loss for each datum. Finding the
74 best weights w, b now boils down to solving the optimization problem

$$w^*, b^* = \underset{w \in \mathbb{R}^d, b \in \mathbb{R}}{\operatorname{argmin}} \ell(w, b) \quad (2.3)$$

75 **How to solve the optimization problem?** We will learn many techniques to
76 solve problems of the form (2.3). We have a simple case here and therefore
77 can use what you did in HW0. The solution is given by

$$w^* = (\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}})^{-1} \tilde{\mathbf{X}}^\top \mathbf{Y} \quad (2.4)$$

78 where we have denoted by $\tilde{\mathbf{X}} \in \mathbb{R}^{n \times (d+1)}$ the matrix whose i^{th} row is the
79 datum with a constant entry 1 appended at the end $[x^i, 1]$. Similarly $\mathbf{Y} \in \mathbb{R}^n$
80 is a vector whose i^{th} entry is the target y^i .

$$z = [x_1, \dots, x_d, 1, x_1 x_2, x_2 x_3, \dots, \dots, x_1 x_2 x_3, \dots, \dots]$$

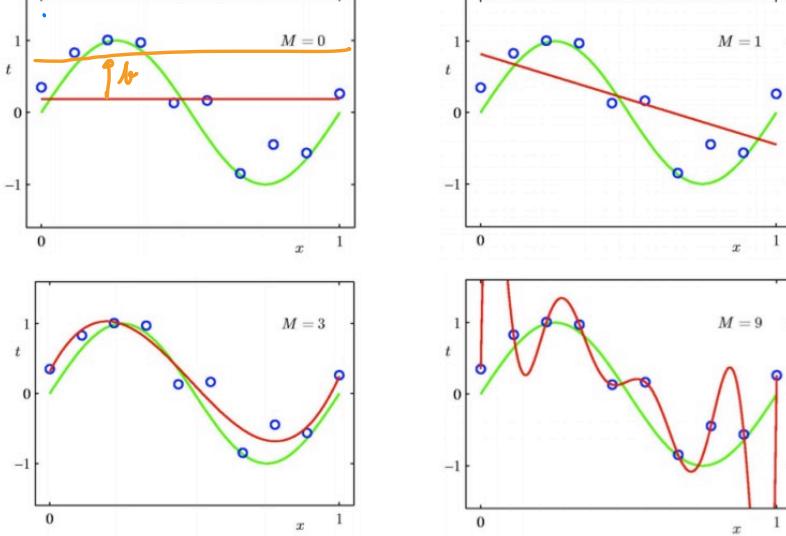


Figure 2.2: Least squares fitting using polynomials. As the degree of the polynomial M increases the predictor f fits the training data (in blue) better and better. But such a well-fitted predictor may be very different from the true model from which Nature generated the data (in green). The red curve in the fourth panel in these cases is said to have been *over-fitted*.

81 2.2.1 Maximum Likelihood Estimation

82 There is another perspective to fitting a machine learning model. We will
83 suppose that our training data was created using a statistical model. We write
84 this as

$$y = w^\top x + b + \epsilon \quad (2.5)$$

7 Why use the average, as opposed
8 to say the maximum value?

Condition:

① $n > d+1$

② Full-rank of $\tilde{\mathbf{X}}^\top \tilde{\mathbf{X}}$

7 When is our solution to least
8 squares regression in (2.4) not
9 defined?

7 What are we losing by fitting a
8 linear predictor? Will this work if
9 the true model from which Nature
10 generates the data was different, say
11 a polynomial?

$$z = [x, 1, x^2, \dots, x^d]$$

$$w = [w_1, w_2, \dots, w_d]$$

$$\mathcal{L}(w) = \sum_{i=1}^d w_i^2$$

$$\ell(w, b) + \mathcal{L}(w)$$

85 Of course we do not know whether Nature used this particular model $f(x; w, b) :=$
 86 $w^\top + b$ to create the data, it might have created the data using some other
 87 model. This discrepancy between the models is *modeled* as noise ϵ . Noise in
 88 machine learning comes from the fact that we the user do not know Nature's
 89 model.

90 What model is appropriate for the noise ϵ ? There can be many models
 91 depending upon your experiment (think of a model that predicts the arrival
 92 time of a bus at the bus stop, what noise would you use?). For our purpose we
 93 will use zero-mean Gaussian noise

$$\epsilon \sim N(0, \sigma_\epsilon^2)$$

94 that does not depend on the sample x . The probability that a sample (x^i, y^i)
 95 in our dataset D_{train} was created using our statistical model is then

$$p(y^i | x^i, w, b) = N(w^\top x^i + b, \sigma_\epsilon^2).$$

96 We have assumed that the data was drawn iid by Nature so the likelihood of
 97 our entire dataset is

$$p(D_{\text{train}} | w, b) = \prod_{i=1}^n p(y^i | x^i, w, b).$$

98 Finding good values of w, b can now be thought of as finding values that
 99 maximize the likelihood of our observed data

$$w^*, b^* = \underset{w, b}{\operatorname{argmin}} -\log p(D_{\text{train}} | w, b). \quad \text{p}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{\sigma^2}} \quad (2.6)$$

100 Observe that our objective is written as the minimization of the *negative log-likelihood*. This is equivalent to maximizing the likelihood because logarithm
 101 is monotonic function. We can now rewrite the objective as

$$-\log p(D_{\text{train}} | w, b) = \frac{n}{2} \log(\sigma_\epsilon^2) + \frac{n}{2} \log(2\pi) + \frac{1}{2\sigma_\epsilon^2} \sum_{i=1}^n (y^i - w^\top x^i - b)^2.$$

103 Notice that only the third term depends on w, b . The first term is a function
 104 of our *chosen* value σ_ϵ^2 , the second term is a constant. In other words, finding
 105 maximizing the likelihood boils down to solving the optimization problem

$$w^*, b^* = \underset{w, b}{\operatorname{argmin}} \frac{1}{2\sigma_\epsilon^2} \sum_{i=1}^n (y^i - w^\top x^i - b)^2. \quad (2.7)$$

106 This objective is nothing other than our least squares regression objective with
 107 σ_ϵ^2 set to 1. This objective known as the maximum likelihood objective (MLE).

109 Maximum likelihood objective has an interesting offshoot. In the least
 110 squares case, given an input x all that our fitted model could predict was

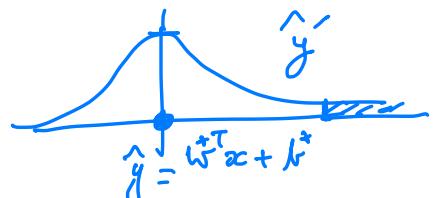
$$\hat{y} = w^*^\top x + b^*.$$

111 MLE has helped us fit a statistical model to the data. So we can now predict
 112 the entire distribution

$$p(y | x, w^*, b^*) = N(w^*^\top x + b^*, \sigma_\epsilon^2).$$

?

Can you think any other sources of noise? For instance, if you scraped some images from the Internet, how will you label them?



113 The solution of least squares is the mean of the Gaussian random variable
 114 $y|x, w^*, b^*$, the variance of this random variable is σ_ϵ^2 . So instead of just pre-
 115 dicting \hat{y} the machine learning model can now give the probability distribution
 116 $p(y|x, w^*, b^*)$ as the output and the user is free to use it as they wish, e.g.,
 117 compute the mean, the median, the 5% probability value of the right tail etc.

118 2.3 Perceptron

119 Let us now solve a classification problem. We will again go around the model
 120 selection problem and consider the class of linear classifiers. Assume binary
 121 labels $Y \in \{-1, 1\}$. To keep the notation clear, we will use the trick of
 122 appending a 1 to the data x and hide the bias term b in the linear classifier. The
 123 predictor is now given by

$$f(x; w) = \text{sign}(w^\top x) \\ = \begin{cases} +1 & \text{if } w^\top x \geq 0 \\ -1 & \text{else.} \end{cases} \quad (2.8)$$

124 We have used the sign function denoted as sign to get binary $\{-1, +1\}$ outputs
 125 from our real-valued prediction $w^\top x$. This is the famous perceptron model of
 126 Frank Rosenblatt. We can visualize the perceptron the same way as we did for
 127 linear regression.

128 Let us now formulate an objective to fit/train the perceptron. As usual, we
 129 want the predictions of the model to match those in the training data.

$$\ell_{\text{zero-one}}(w) := \frac{1}{n} \sum_{i=1}^n \mathbf{1}_{\{y^i \neq f(x^i; w)\}}. \quad (2.9)$$

130 The indicator function inside the summation measures the number of mistakes
 131 the perceptron makes on the training dataset. The objective here is designed to
 132 find weights w that minimizes the average number of mistakes, also known as
 133 the training error. Such a loss that measures the mistakes is called the zero-one
 134 loss, it incurs a penalty of 1 for a mistake and zero otherwise.

135 2.3.1 Surrogate Losses

136 The zero-one loss is the clearest indication of whether the perceptron is work-
 137 ing well. It is however non-differentiable, so we cannot use powerful ideas
 138 from optimization theory to minimize it. This is why surrogate losses are con-
 139 structed in machine learning. These are proxies for the loss function, typically
 140 for the classification problems and look as follows.

141 The hinge loss is one such surrogate loss. It is given by

$$\ell_{\text{hinge}}(w) = \max(0, -y w^\top x).$$

142 If the predicted label $\hat{y} = \text{sign}(w^\top x)$ have the same sign as the true label y ,
 143 the hinge-loss is zero. If they have opposite signs, the hinge loss increases
 144 linearly. The exponential loss

$$\ell_{\text{exp}}(w) = e^{-y (w^\top x)}$$

?

Is a linear model appropriate if our data was natural images? What properties have we lost by restricting the classifier to be linear?

▲

The linear classifier remains unchanged if we reorder the pixels of all images consistently in our entire training set and the weights w . The images will look nothing like real images to us. The perceptron does not care about which pixels in the input are close to which others.

?

Can you think of some quantity other than the zero-one error that we may wish to optimize?

¹⁴⁵ or the logistic loss

$$\ell_{\text{logistic}}(w) = \log \left(1 + e^{-y w^\top x} \right)$$

¹⁴⁶ are some other popular losses for classification.

❸ Draw the three losses to observe their differences.

¹⁴⁷ 2.4 Stochastic Gradient Descent

¹⁴⁸ We will now fit a perceptron using the hinge loss using a very simple optimization
¹⁴⁹ technique. At each iteration, this algorithm updates the weights w in the
¹⁵⁰ direction of the negative gradient. So first, let us compute the gradient of the
¹⁵¹ hinge loss. It is easily seen to be

$$\frac{d\ell_{\text{hinge}}(w)}{dw} = \begin{cases} -y x & \text{for incorrect prediction} \\ 0 & \text{else.} \end{cases} \quad (2.10)$$

¹⁵² We will use a naive algorithm to update the weights. Here is how it goes.

The Perceptron algorithm

Perform the following steps for iterations $t = 1, 2, \dots$

1. At the t^{th} iteration, sample a data point from D_{train} uniformly randomly, call it (x^t, y^t) .
2. Update the weights of the perceptron as

$$w_{t+1} = \begin{cases} w_t + y^t x^t & \text{if } \text{sign}(w_t^\top x^t) \neq y^t \\ w_t & \text{else.} \end{cases} \quad (2.11)$$

¹⁵³ In other words, the perceptron weights is changed only if it makes a
¹⁵⁴ mistake on the sample (x^t, y^t) . The updated perceptron improves its mistake
¹⁵⁵ on this sample. Observe that a mistake happens if the sign of $w^\top x^t$ and y^t are
¹⁵⁶ different, the product $y^t w^\top x^t$ is therefore negative. The updated weights of
¹⁵⁷ the perceptron now satisfy

$$\begin{aligned} y^t (w_t + y^t x^t)^\top x^t &= y^t w^\top x^t + y^t x^\top x^t \\ &= y^t w^\top x^t + \|x^t\|_2^2. \end{aligned}$$

¹⁵⁸ In simple worlds, the value of $y^t w^\top x^t$ increases as a result of the update,
¹⁵⁹ it becomes more positive. If the perceptron makes mistakes on the same
¹⁶⁰ datum repeatedly, this value is eventually going to become positive. Of course,
¹⁶¹ mistakes on other data in the training set may steer the perceptron towards
¹⁶² other directions and it may continue to cycle ad infinitum, but it is easy to show
¹⁶³ that it ceases its updates when all data are correctly classified. More precisely,
¹⁶⁴ if the training data are such that they can be correctly classified using a linear
¹⁶⁵ predictor, then the perceptron will find this predictor after a finite number of
¹⁶⁶ iterations.

167 We have seen a powerful algorithm for machine learning. This algorithm is
168 called stochastic gradient descent (SGD) and it is very general: so long as you
169 can take the gradient of the objective you can execute SGD. The algorithm for
170 fitting the perceptron above was given by Rosenblatt in 1957 and is popularly
171 known as the “perceptron algorithm”. It is interesting to note that it is simply
172 the instantiation of SGD which was known before [Robbins and Monro \(1951\)](#)
173 for the hinge loss.

?

You may have seen the hinge loss written as
 $\ell_{\text{hinge}}(w) = \max(0, 1 - y w^\top x)$.
Why the difference?

¹⁷⁴ Bibliography

- ¹⁷⁵ Rahimi, A. and Recht, B. (2008). Random features for large-scale kernel
¹⁷⁶ machines. In *Advances in neural information processing systems*, pages
¹⁷⁷ 1177–1184.
- ¹⁷⁸ Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The
annals of mathematical statistics*, pages 400–407.