LECTURE 2 - REVIEW OF LINEAR ALGEBRA AND INTRODUCTION TO OVERFITTING & UNDERFITTING

1. Polynomial Curve Fitting Example

- Suppose we are given a training set comprising of N observations of x, $\mathbf{x} = (x_1, x_2, \ldots, x_N)^T$, along with desired outputs $\mathbf{t} = (t_1, t_2, \ldots, t_N)^T$
- We generally organize data into *vectors* and *matrices*. Not only is it a common way to organize the data, but it allows us to easily apply linear algebraic operations during analysis.
- Let us review some linear algebra:
  - What is a vector?

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_D \end{bmatrix} \tag{1}$$

  - What is the transpose operation?

$$\mathbf{x}^T = [x_1, x_2, \cdots, x_D] \tag{2}$$

  - Vectors are often used to represent a data point (as a concatenation of all of the associated features for the data point)
  - Given a vector $\mathbf{x} \in \mathbb{R}^D$ and a scalar value a, what is $a\mathbf{x}$? What does this operation do geometrically?
  - Given $\mathbf{x} \in \mathbb{R}^D$ and $\mathbf{y} \in \mathbb{R}^D$, what is $\mathbf{x} + \mathbf{y}$? What is the geometric interpretation?
  - Given $\mathbf{x} \in \mathbb{R}^D$ and $\mathbf{y} \in \mathbb{R}^D$, what is $\mathbf{x} - \mathbf{y}$?
  - An important operation is the *inner product*:

$$\mathbf{x}^T\mathbf{y} = \mathbf{y}^T\mathbf{x} = \sum_{i=1}^{D} x_i y_i \tag{3}$$

  - What is the *outer product*?

$$xy^\top = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}^\top = \begin{bmatrix} x_1y_1 & x_1y_2 & \cdots & x_1y_n \\ x_2y_1 & x_2y_2 & \cdots & x_2y_n \\ \vdots & \vdots & \ddots & \vdots \\ x_dy_1 & x_dy_2 & \cdots & x_dy_n \end{bmatrix} \in \mathcal{R}^{d \times n}. \tag{4}$$

- Note the notation: scalar values are unbolded (e.g., N, x), vectors are lower case and bolded (e.g., $\mathbf{x}$), arrays are uppercase and bolded (e.g., $A \in \mathbb{R}^{d \times N}$), vectors are generally assumed to be column vectors (e.g., $\mathbf{x}^T = (x_1, \ldots, x_N)$ and $\mathbf{x} = (x_1, \ldots, x_N)^T$)
- Suppose the data truly came from some unknown hidden function. All we have is the training data (which includes noise) and we want to learn a mapping from input values x to desired output values t. We will learn (i.e., train a model to estimate) that mapping from the training data $\{\mathbf{x}, \mathbf{t}\}$. Then, when we are given test data, we can predict each test data point's t value.
- For this problem, we assume that the original data x is sufficient and appropriate (so, we do not need to preprocess or extract features). Then, we have completed steps 1 and 2 of the general approach listed in Section 1 above.
- Now we must assume a model. Lets assume a polynomial function as our model:

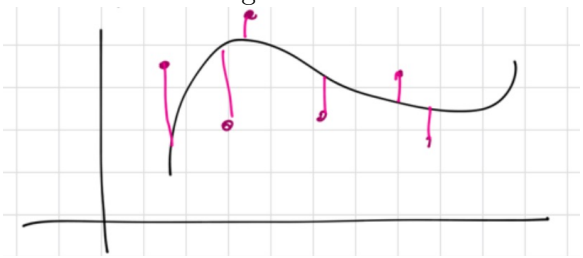
$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j \tag{5}$$

- Now we must *train* this model by estimating the unknown parameters ($\mathbf{w}$) that maps the training data, $\mathbf{x}$, to their desired values, $\mathbf{t}$, given some assumed value for M
- So, we have N discrete points from which to estimate $\mathbf{w}$. We can minimize the squared error to estimate the parameters:

$$\arg\min_{\mathbf{w}} E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} (y(x_n, \mathbf{w}) - t_n)^2 \tag{6}$$

$$= \frac{1}{2} \sum_{n=1}^{N} \left(\sum_{j=0}^{M} w_j x_n^j - t_n \right)^2 \tag{7}$$

- Consider the following illustration of the error function:



- We can write the error function compactly in matrix/vector form:

$$E(\mathbf{w}) = \frac{1}{2}\left([w_0, w_1, \ldots, w_M]\begin{bmatrix} 1 & 1 & \ldots & 1 \\ x_1 & x_2 & \ldots & x_n \\ x_1^2 & x_2^2 & \ldots & x_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^M & x_2^M & \ldots & x_n^M \end{bmatrix} - [t_1, t_2, \ldots, t_N]\right)$$
$$\left([w_0, w_1, \ldots, w_M]\begin{bmatrix} 1 & 1 & \ldots & 1 \\ x_1 & x_2 & \ldots & x_n \\ x_1^2 & x_2^2 & \ldots & x_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^M & x_2^M & \ldots & x_n^M \end{bmatrix} - [t_1, t_2, \ldots, t_N]\right)^T$$

- So, we want $E(\mathbf{w})$ to be small. How do we solve for $\mathbf{w}$?
- We can take the derivative of the error function, set it to zero, and solve for the parameters. In general, this method does not guarantee that the parameters we estimate are minima of the error function (e.g., may be an inflection point, maxima). It is a necessary condition (but not sufficient). However, if the function is convex, then it will always find the global optima.
- How do we take the derivative of a function with respect to a vector?

$$\frac{\partial}{\partial \mathbf{x}} f(\mathbf{x}) = \left[\frac{\partial}{\partial x_1} f(\mathbf{x}), \frac{\partial}{\partial x_2} f(\mathbf{x}), \ldots, \frac{\partial}{\partial x_n} f(\mathbf{x})\right]^\top \in \mathcal{R}^{n \times 1}.$$

- So, what would the derivative of $E(\mathbf{w})$ be with respect to $\mathbf{w}$?

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(\sum_{j=0}^{M} w_j x_n^j - t_n\right)^2 \tag{8}$$

$$\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} = \left[\frac{\partial E(\mathbf{w})}{\partial w_0}, \frac{\partial E(\mathbf{w})}{\partial w_1}, \ldots, \frac{\partial E(\mathbf{w})}{\partial w_M}\right]^T \tag{9}$$

$$= \left[\sum_{n=1}^{N}\left(\sum_{j=0}^{M} w_j x_n^j - t_n\right) x_n^0, \sum_{n=1}^{N}\left(\sum_{j=0}^{M} w_j x_n^j - t_n\right) x_n^1, \ldots, \sum_{n=1}^{N}\left(\sum_{j=0}^{M} w_j x_n^j - t_n\right) x_n^M\right]^T \tag{10}$$

Similarly,

$$\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}} = \frac{1}{2} 2\mathbf{X}^T\left(\mathbf{w}^T\mathbf{X}^T - \mathbf{t}^T\right)^T \tag{11}$$

where $\mathbf{X}^T = \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_n \\ x_1^2 & x_2^2 & \dots & x_n^2 \\ \vdots & \vdots & \ddots & \vdots \\ x_1^M & x_2^M & \dots & x_n^M \end{bmatrix}$.

- Then, we can set the derivative to zero and solve:

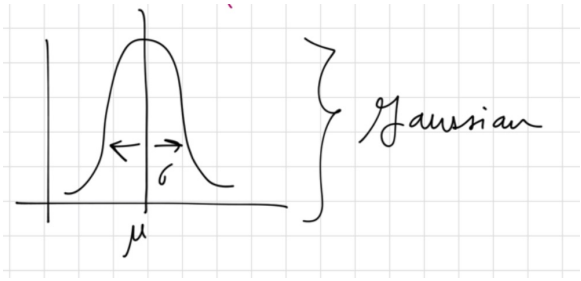
$$0 = \mathbf{X}^T\mathbf{X}\mathbf{w} - \mathbf{X}^T\mathbf{t} \tag{12}$$

$$\mathbf{X}^t\mathbf{t} = \mathbf{X}^T\mathbf{X}\mathbf{w} \tag{13}$$

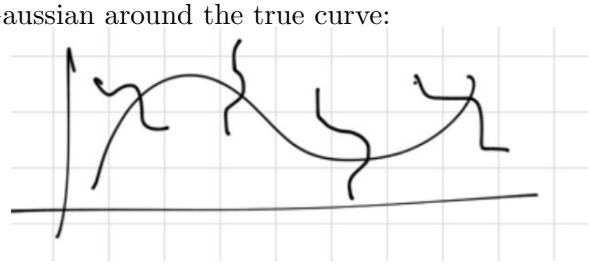
$$\mathbf{w} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{t} \tag{14}$$

- Suppose our data actually came from: $t = \sin(2\pi x) + \epsilon$ where ϵ is Gaussian zero-mean random noise.
- The univariate Gaussian Distribution:

$$\mathcal{N}(x|\mu, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\} \tag{15}$$



- The the noise is zero-mean Gaussian distributed, it is like we are saying there is a Gaussian around the true curve:



$$t = y + \epsilon \tag{16}$$

$$\epsilon = t - y \tag{17}$$

where

$$\epsilon \sim \mathcal{N}(0, \sigma^2) \tag{18}$$

thus

$$\mathcal{N}(t-y|0,1) \propto \exp\left\{-\frac{1}{2}\frac{(t-y-0)^2}{1^2}\right\} \tag{19}$$

$$= \exp\left\{-\frac{1}{2}(t-y)^2\right\} \tag{20}$$

$$= \exp\{-E(\mathbf{w})\} \tag{21}$$

- So, the squared error objective function, $E(\mathbf{w})$, assumes Gaussian noise.
- Another way to look at it: t is distributed according to a Gaussian distribution with mean y

2. Overfitting/Overtraining

- In the polynomial curve fitting example, M is the *model order*. As M increases, there are more parameters (more w) to learn and, so, the model becomes more complex. As a model is more and more complex, it is more likely to *overfit* or *overtrain*. This essentially means it may "memorize" the input training data (including all of the training data's noise). Overfitting means that the performance of the model will likely decrease on unknown test data. Overfitting means that the "true" underlying model of the data is not estimated/learned but instead results in a poor representation that memorizes meaningless noise in the data.
- There are two common approaches to avoid overfitting:
  (1) More data: As you have more and more data, it becomes more and more difficult to "memorize" the data and its noise. Often, more data translates to the ability to use a more complex model and avoid overfitting. However, generally, you need exponentially more data with increases to model complexity. So, there is a limit to how much this helps. If you have a very complex model, you need a huge training data set.
  (2) Regularization: Regularization methods add a penalty term to the error function to discourage overfitting. These penalty terms encourage small values limiting the ability to overfit. (This is just a teaser. We will discuss this further in the future.)
- You can also *underfit* your data. When you underfit, you model complexity is not complex enough to model all of the complexities in your data.

3. Cross Validation and Training, Validation, Testing Data Sets

- Due to the potential for overfitting, performance on a training set is not a good indicator of general performance.
- One approach to estimate performance of a system is to use of the data for training and some for validation.
- If model design is iterated many times, you may even over fit the validation set. So, you can hold out another set for testing.

- Given limited data, we want to use any much data for training as possible. But this may leave us with too little validation or testing data. One method to help alleviate this issue is cross-validation.
- However there are downsides to cross-validation: need to train many times (which can sometimes be very computationally complex), you end up with several models - how do you pick the final one to use?

4. Error and Accuracy Metrics

- When training or doing cross-validation, you rely on an evaluation metric.
- There are many standard ones used
- However, before working on a problem it is best to think carefully about how to evaluate your results - it has a big impact on the effectiveness of your final method