Unit 4.1 – Interpolation

Interpolation is the idea that you can predict the value of data between two (or more) known data points.

Extrapolation is the idea that you can predict the values of data outside of two (or more) know data points.

Both ideas presuppose that a functional relationship describes the data.

Consider the equation for a line found in Unit 2.1

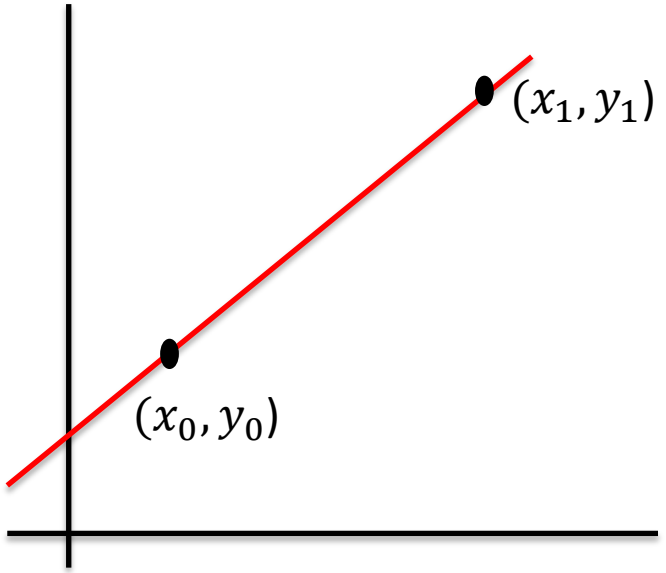
$$ax + by = c \qquad (2.1\text{-}6)$$

Rewritten

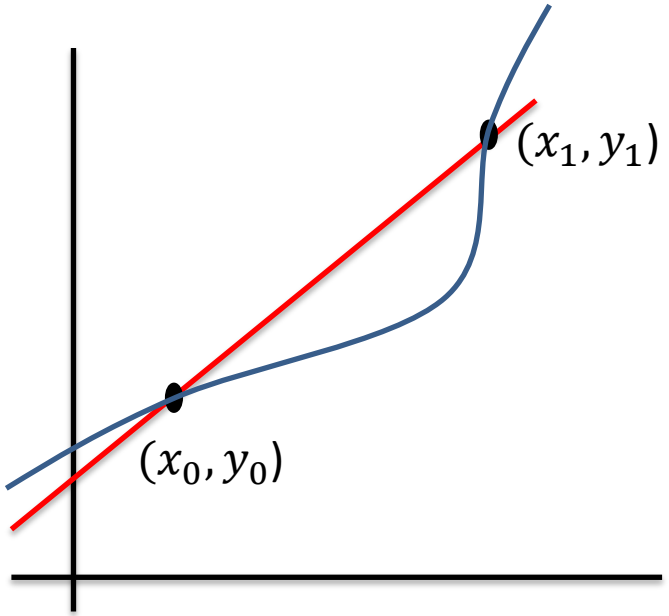
$$\frac{a}{c}x + \frac{b}{c}y = 1$$

$$dx + ey = 1 \qquad (4.1\text{-}1)$$

Showing it is really one equation in 2 unknowns. So that given points (x_0, y_0) and (x_1, y_1) we can determine *d* and *e.* Once known, a point anywhere in between can be identified simply by using Eq. 4.1-1.



But what if the function really looks like this?



To some degree of accuracy, we can use linear interpolation provided we have enough data points to approximate the curvature with lines. If we don't, we need something more sophisticated. Knowledge of the functional form is always preferred but might not actually be available.

Alternatively, we can use the extra data to define higher-order functions.

For *n* data points, we can employ polynomials up to order *n-1*.

Lagrange's Method

This approach defines the interpolating polynomial as

$$P_n(x) = \sum_{i=0}^{n} y_i l_i(x) \qquad (4.1\text{-}2)$$

Where:

$$l_i(x) = \frac{x-x_0}{x_i-x_0} \cdot \frac{x-x_1}{x_i-x_1} \cdots \frac{x-x_n}{x_i-x_n}$$

$$= \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j}, \qquad i = 0,1,\cdots,n \qquad (4.1\text{-}3)$$

Are the cardinal functions or weights.

For our linear example, the weights are:

$$l_0(x) = \frac{x-x_1}{x_0-x_1} \qquad l_1(x) = \frac{x-x_0}{x_1-x_0} \quad (4.1\text{-}4)$$

The interpolating polynomial is then:

$$P_1(x) = y_0 l_0(x) + y_1 l_1(x) \quad (4.1\text{-}5)$$

The $l_i(x)$ tell us how much of each y_i we should use.

With 3 points we can use an *n=2* polynomial.

$$P_2(x) = y_0 l_0(x) + y_1 l_1(x) + y_2 l_2(x) \quad (4.1\text{-}6)$$

Where:

$$l_0(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)} \quad (4.1\text{-}7)$$

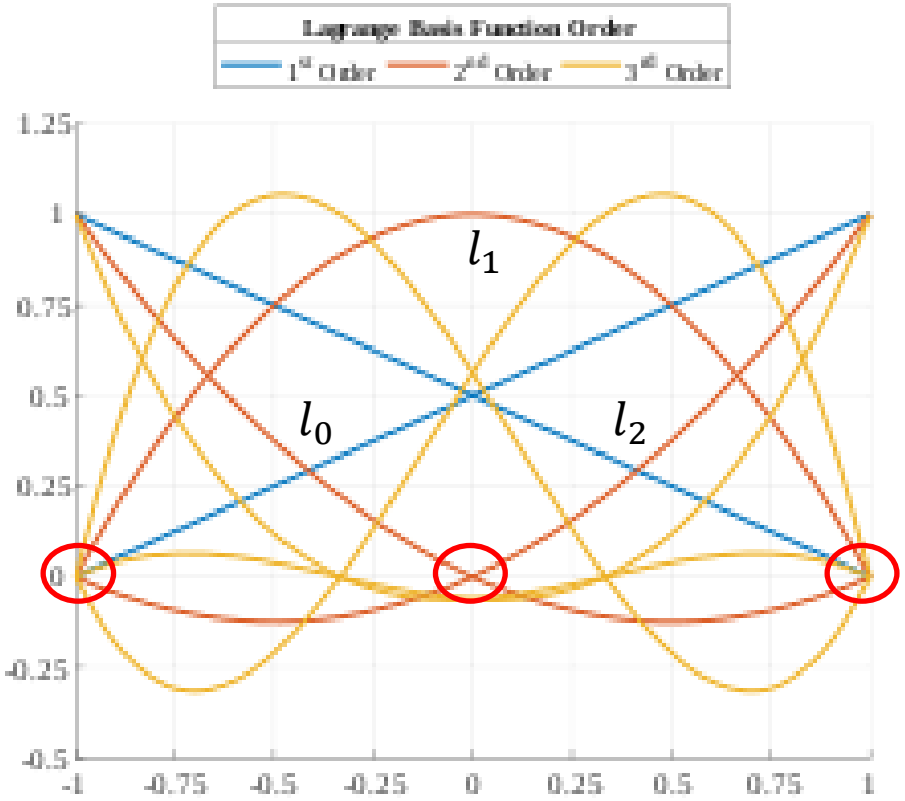
$$l_1(x) = \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}$$

$$l_2(x) = \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}$$

The weights have the property

$$l_i(x_j) = \begin{Bmatrix} 0 \; if \; i \neq j \\ 1 \; if \; i = j \end{Bmatrix} = \delta_{ij}$$

For $x_0 = -1$, $x_1 = 0$, and $x_2 = 1$ the 2nd order cardinal functions are shown in orange below.



A linear example problem from Gas Dynamics:

Consider the isentropic flow tables. Properties like $\frac{p_0}{p}$ are plotted against Mach number. How accurate is a point between? Note that the actual equation is:

$$\frac{p_0}{p} = \left(1 + \frac{\gamma-1}{2} M^2\right)^{\gamma/\gamma-1} \qquad (4.1\text{-}8)$$

M	p_o/p – table	p_o/p – (4.1 – 8)
0.20	1.028	1.02828112112
0.22	1.034	1.03429193455
0.23	1.038	1.03752231580
0.24	1.041	1.04090395711
0.26	1.048	1.04812512315

How accurate is an interpolated value for M=0.23?

Using Eqs. 4.1-4 & 4.1-5:

$$P_1(0.23) = 1.034\frac{0.23-0.24}{0.22-0.24} + 1.041\frac{0.23-0.22}{0.24-0.22}$$

$$\frac{p_0}{p} = P_1(0.23) = 1.0375$$

Which should be compared to the Eq. 4.1-8 value of 1.03752231580. In other words, 5-digit accuracy is produced. However, using Eq. 4.1-8 values at Mach 0.22 and 0.24 produces 1.03759794583, which is technically only 4-digit accuracy, equal to the table. Linear interpolation is therefore within the accuracy of the table.

The reason for the "better" prediction with the rounded numbers is a bit of a fluke, since 0.22 is slightly lower and 0.24 slightly higher than the actual.

This begs the question can we get a more accurate interpolation?

Let's try with P_3, using 4 table points

$$P_3(x) = y_0 l_0(x) + y_1 l_1(x) + y_2 l_2(x) + y_3 l_3(x)$$

$$l_0(x) = \frac{(x-x_1)(x-x_2)(x-x_3)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)} = -0.0625$$

$$l_1(x) = \frac{(x-x_0)(x-x_2)(x-x_3)}{(x_1-x_0)(x_1-x_2)(x_1-x_3)} = 0.5625$$

$$l_2(x) = \frac{(x-x_0)(x-x_1)(x-x_3)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)} = 0.5625$$

$$l_3(x) = \frac{(x-x_0)(x-x_1)(x-x_2)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)} = -0.0625$$

If we use the table values:

$$P_3(0.23) = 1.0374375$$

Interpolated equation 4.1-8 values:

$$P_3(0.23) = 1.03752229879$$

Compared to the actual Eq.. 4.1-8 value:

$$\frac{p_0}{p} = 1.0375223158$$

Demonstrating that the higher order interpolation buys something if higher order data is available.

Keep in mind that for a given set if *n+1* points a unique *n*-degree polynomial exists. Three polynomial interpolation methods are presented in the text

Lagrange – simple presentation.

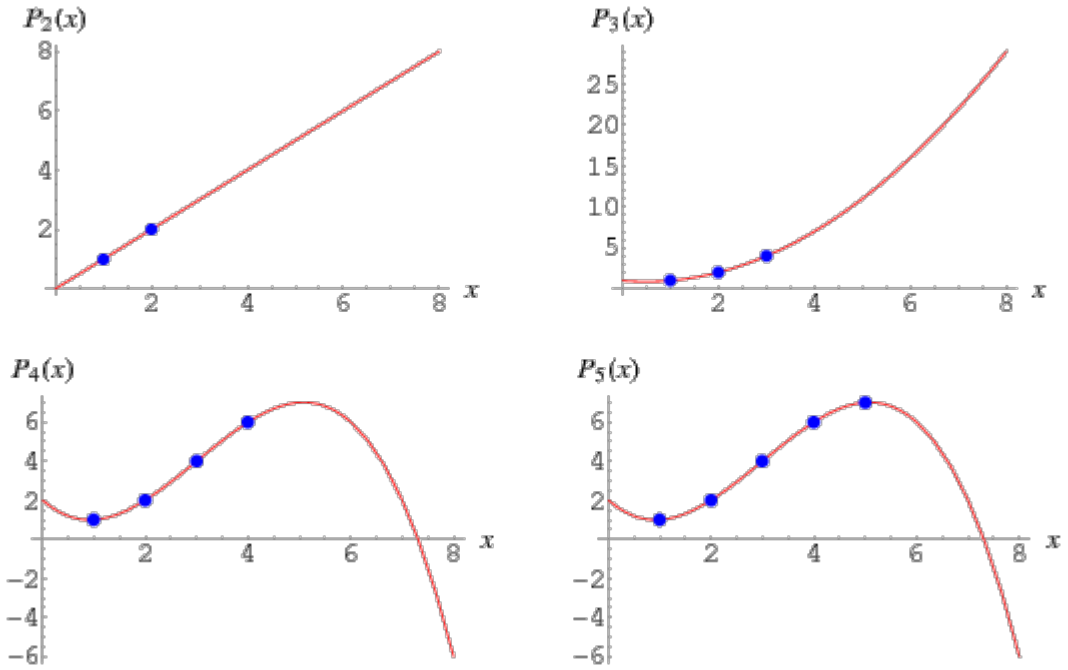
Newton – ease of coding, good if multiple interpolations are being done - newtonPoly

Neville – good if a single interpolation is done - neville

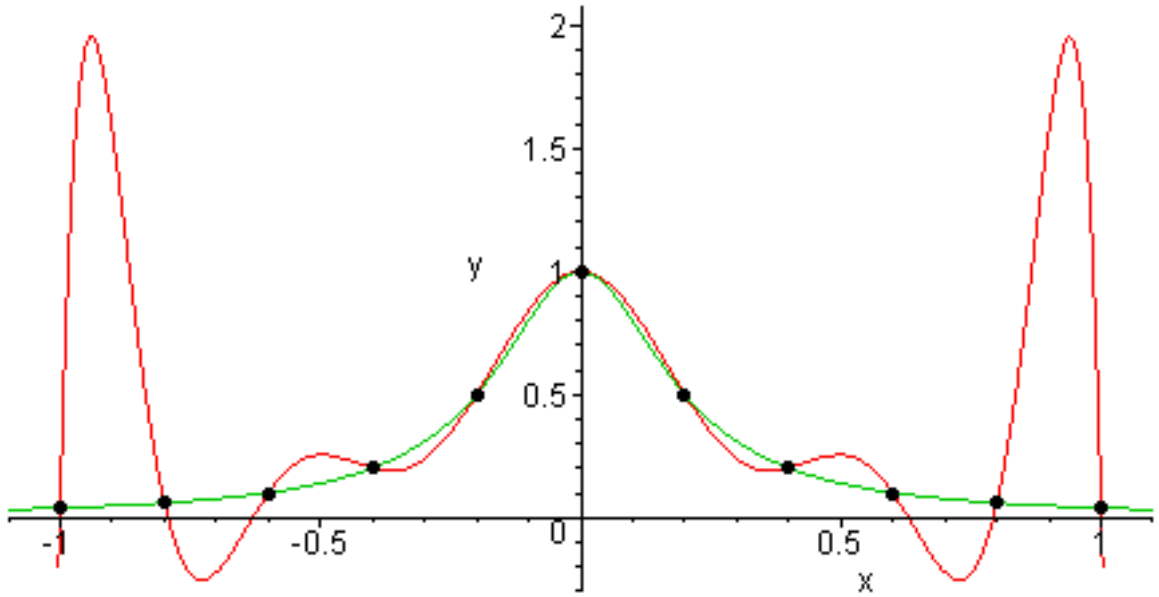
These techniques produce essentially the same result, since a unique polynomial exists for any set of data.

Limits of Polynomial Interpolation

The suggestion is that the higher the polynomial order, the more accurate the interpolation. This image attributed to Wolfram shows the possibilities as n increases.



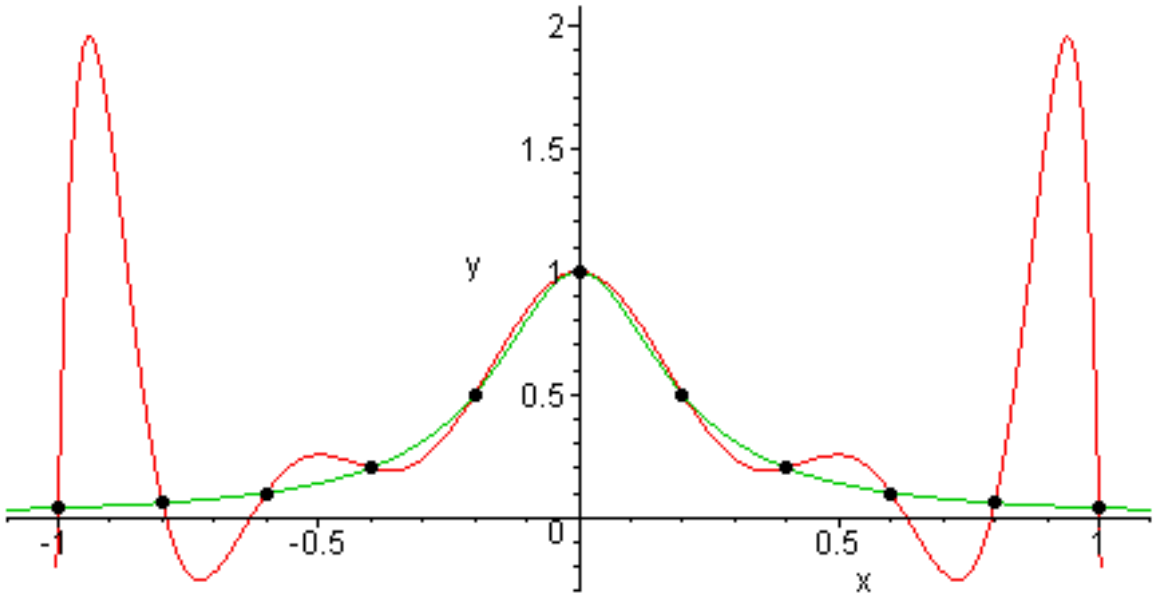
This could be good, as suggested by the Wolfram image, but there is a danger of polynomial overfit, especially if the function is not polynomial in nature. The fit for Runge's function $\frac{1}{25x^2+1}$ shows.

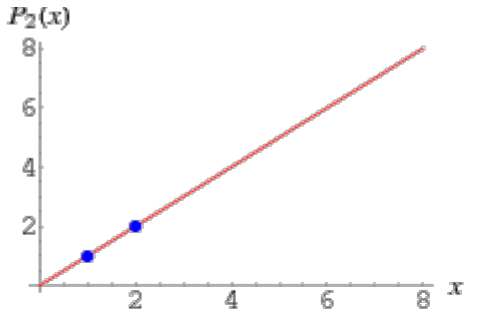


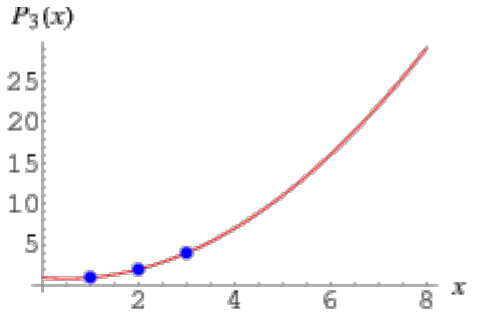
Rules of Thumb – look at the data and choose the lowest order polynomial possible.

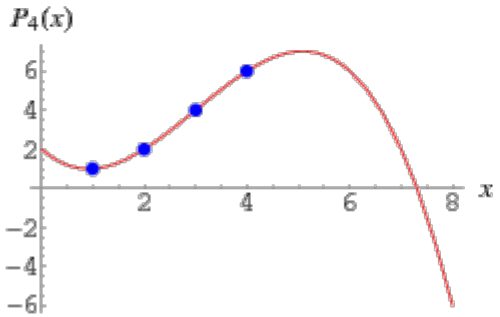
Limits and Extrapolation

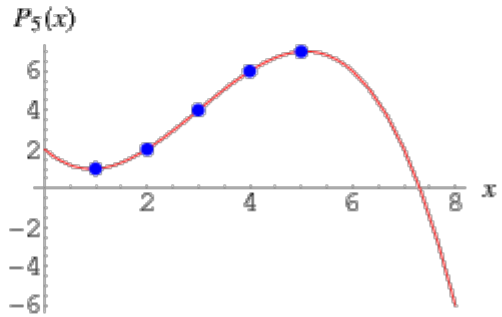
- Higher order captures more variation in the data but can lead to oscillations.
- Best accuracy typically in the center of the interpolating range.
- Extrapolation must be done carefully, especially with higher order polynomials.
- Use of log-scales might improve the interpolation procedure.
- Other functions might be better choices.











Unit 4.2 – Rational Functions and Splines

A rational function is one which can be written:

$$\frac{P(x)}{Q(x)} \qquad (4.2\text{-}1)$$

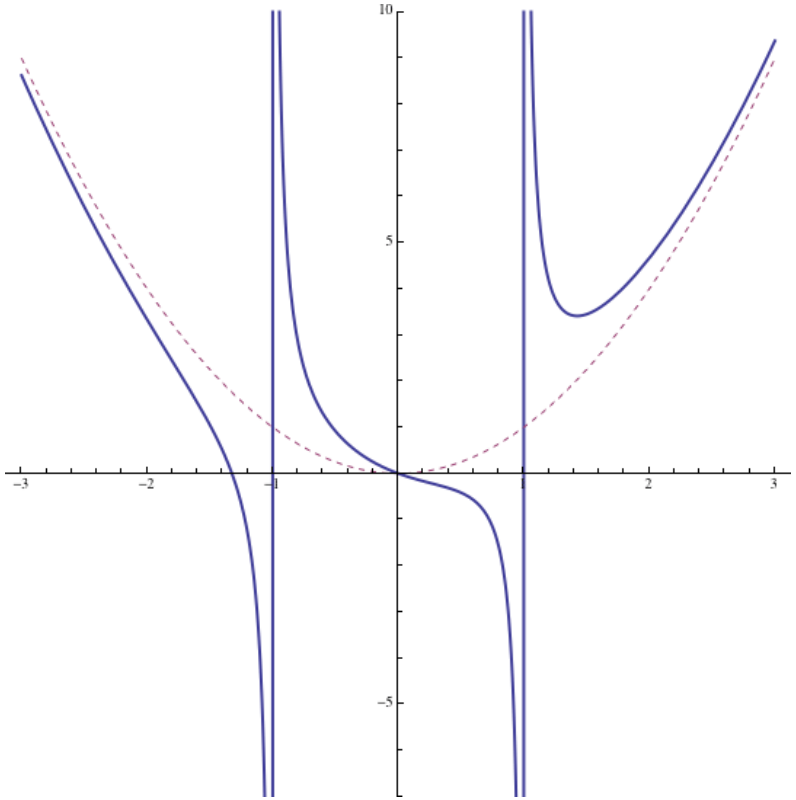
Rational function interpolation usually chooses *P* and *Q* to be polynomials. The order of *P* is either the same as *Q* or one order lower. The text code is *rational*.

Advantages

- Larger range of shapes than polys
- Smoother and less oscillatory
- Better at extrapolation
- Can agree with known asymptotes

Disadvantages

- Not super well understood
- Possible undesirable asymptotes or "poles" can exist near *Q* zeros.



Unit 4.1 – Interpolation

Interpolation is the idea that you can predict the value of data between two (or more) known data points.

Extrapolation is the idea that you can predict the values of data outside of two (or more) know data points.

Both ideas presuppose that a functional relationship describes the data.

Consider the equation for a line found in Unit 2.1

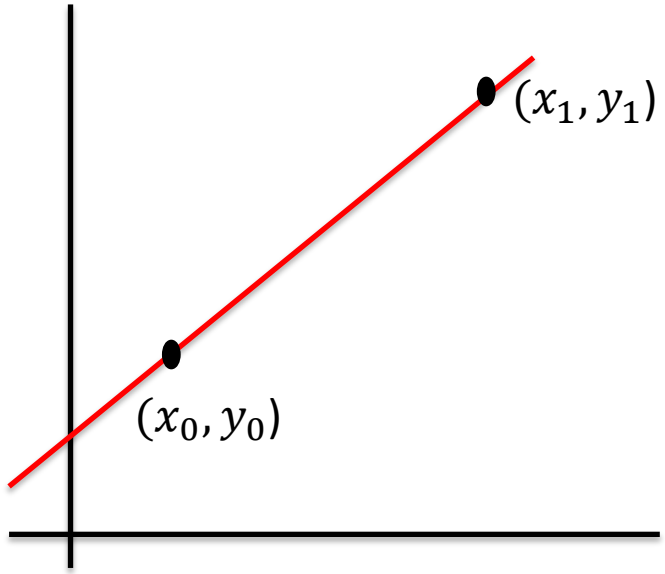
$$ax + by = c \qquad (2.1\text{-}6)$$

Rewritten

$$\frac{a}{c}x + \frac{b}{c}y = 1$$

$$dx + ey = 1 \qquad (4.1\text{-}1)$$

Showing it is really one equation in 2 unknowns. So that given points (x_0, y_0) and (x_1, y_1) we can determine *d* and *e.* Once known, a point anywhere in between can be identified simply by using Eq. 4.1-1.



Unit 4.2 – Rational Functions and Splines

A rational function is one which can be written:

$$\frac{P(x)}{Q(x)} \qquad (4.2\text{-}1)$$

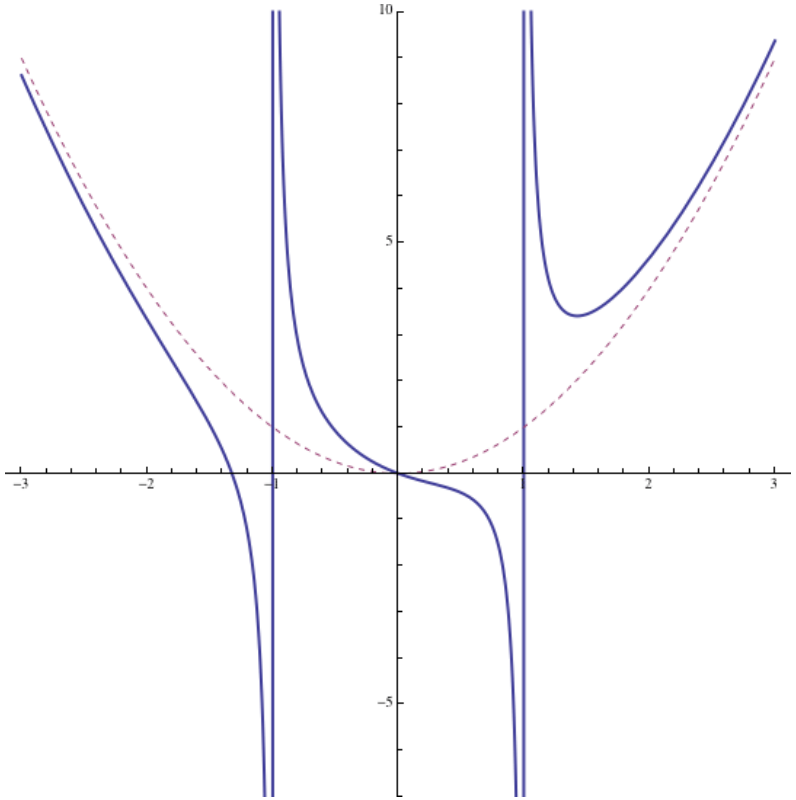
Rational function interpolation usually chooses *P* and *Q* to be polynomials. The order of *P* is either the same as *Q* or one order lower. The text code is *rational*.

Advantages

- Larger range of shapes than polys
- Smoother and less oscillatory
- Better at extrapolation
- Can agree with known asymptotes

Disadvantages

- Not super well understood
- Possible undesirable asymptotes or "poles" can exist near *Q* zeros.



Cubic Splines

An extension to polynomial interpolation is the idea of cubic splines. These are piecewise functions, meaning there is a different function for each segment of data x_i to x_{i+1}.

Recall that the general cubic polynomial

$$P_3(x) = ax^3 + bx^2 + cx + d \quad (4.2\text{-}2)$$

With the 4 unknowns a, b, c, and d.

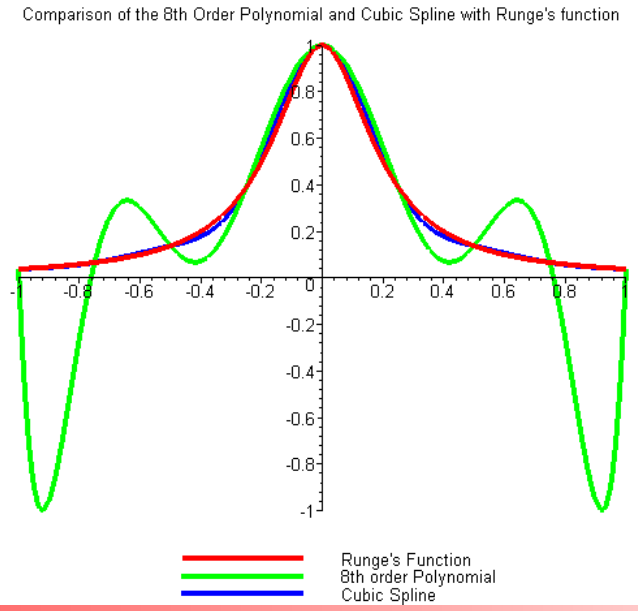
The unknowns are evaluated by:

- 2 – y_i and y_{i+1}
- 1 – continuity of slope at *i.*
- 1 – continuity of curvature at *i.*

With zero curvature for the beginning of the first spline and the end of the last spline.

The code is defined in the text as *cubicSpline*. The curvature and slope continuity conditions produce a tridiagonal matrix for the curvature coefficients, after assuming a linear distribution between the nodes.

Some variants of the technique set the end-point slopes, rather than curvature or deal with the third derivative.



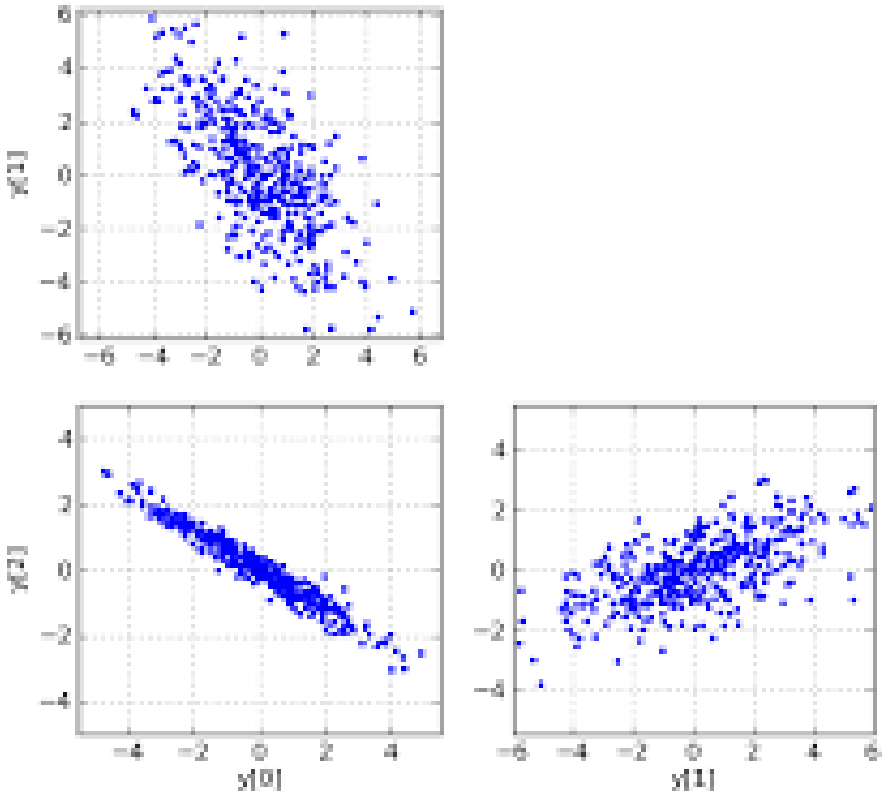
Unit 4.3 – Least Squares

- In Units 4.1 and 4.2 we discussed how to find a function that exactly fits a set of data, that is, the function exactly matches the interpolant at each point. For *n* data points, a unique *n-1* polynomial exists to fit the data.
- We showed the limits of fitting data with high order polynomials.
- All that assumed perfectly correct data – no errors. However, real world systems contain both random and biased data.

Random/Aleatoric uncertainty - repeatability errors caused by unaccounted for factors

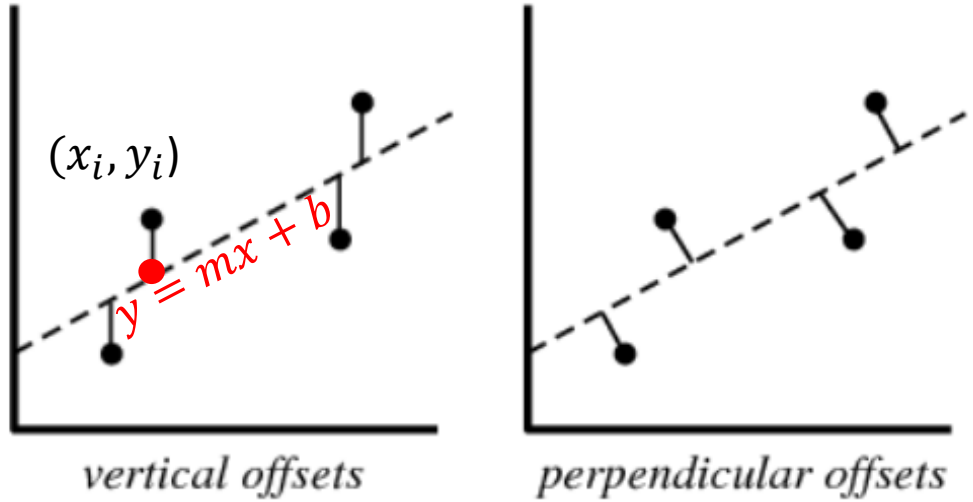
Bias/Epistemic uncertainty – errors in measurement that are repeatable

While we are not entering into the world of uncertainty quantification in this class, we need to understand that data is not pristine or perfectly accurate. Fitting a function to the data will not necessarily describe its behavior. Instead, we might get more from identifying data trends.



How to find the "best" fit

- Let's start by finding the "best" straight line curve.
- "offset" - distance from the line
- Vertical reflects dependent/independent
- Actual vs. squares



- Define: $S \equiv \sum_1^n (y_i - mx_i - b)^2$ (4.3-1)
  - (x_i, y_i) are the data •
  - $y = mx + b$ is the curve, so $mx_i + b$ is the y value of the curve at x_i •
- Choose *m* and *b* to make S as small as possible for all the data.
- σ is the standard deviation defined by $\sigma = \sqrt{\frac{S}{n}}$ for linear fit.

How to get m and b

- We utilize the calculus to minimize S.
- Take the derivatives of S with respect to *m* and *b*, set them to zero, then solve the resulting equations for *m* and *b*.
- This gives:

$$b = \frac{\bar{y}\left(\sum_{i=1}^{n} x_i^2\right) - \bar{x}\left(\sum_{i=1}^{n} x_i y_i\right)}{\sum_{i=1}^{n} x_i^2 - n\bar{x}^2} \qquad (4.3\text{-}2)$$

$$m = \frac{\left(\sum_{i=1}^{n} x_i y_i\right) - n\bar{x}\bar{y}}{\sum_{i=1}^{n} x_i^2 - n\bar{x}^2} \qquad (4.3\text{-}3)$$

- $\bar{x}$ and $\bar{y}$ are the average values of the data points.
- https://mathworld.wolfram.com/LeastSquaresFitting.html

Where:

$\bar{x} = \left(\sum_{i=1}^{n} x_i\right)/n$

$\bar{y} = \left(\sum_{i=1}^{n} y_i\right)/n$

Equation for R^2

UNIVERSITY OF Cincinnati

- Sums of Squares

$$ss_{xx} = \left(\sum_{i=1}^{n} x_i^2\right) - n\bar{x}^2 \quad ; \quad ss_{yy} = \left(\sum_{i=1}^{n} y_i^2\right) - n\bar{y}^2 \quad ; \quad ss_{xy} = \left(\sum_{i=1}^{n} x_i y_i\right) - n\bar{x}\bar{y}$$

- Correlation Coefficient of Determination, R^2

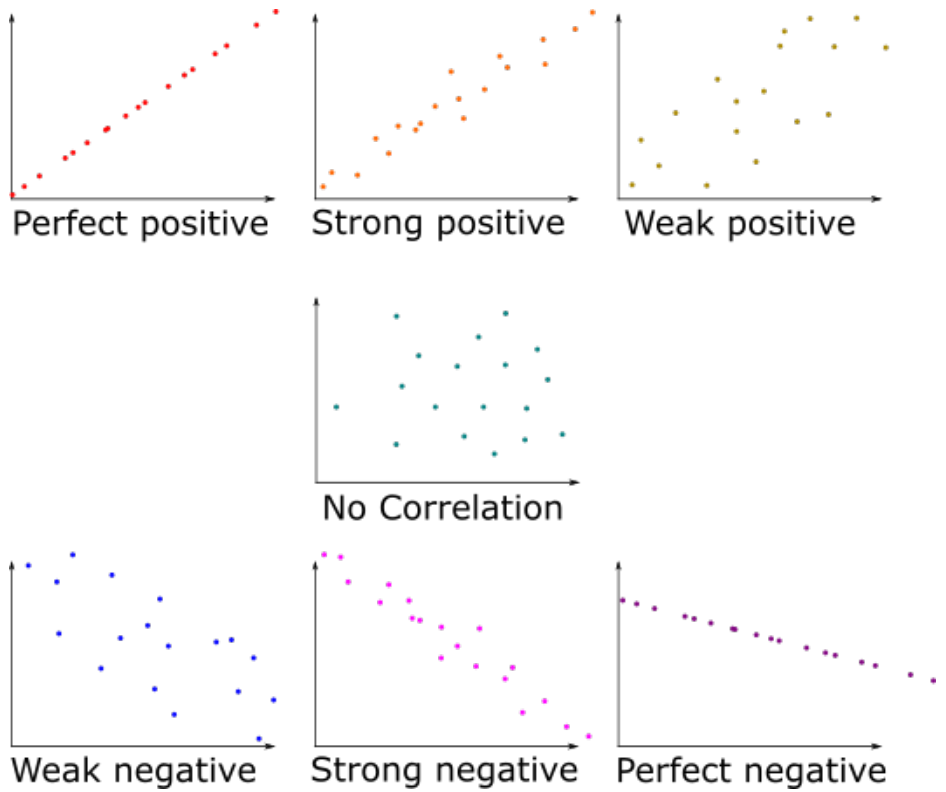
$$R^2 = \frac{ss_{xy}^2}{ss_{xx} ss_{yy}}$$

- R^2 varies between 0 and 1, it provides the percentage variation in *y* explained by the *x* variables

Interpreting the Correlation Coefficient, *R*

Correlation Coefficient	Interpretation
0.9 to 1.0	Very high positive
0.7 to 0.9	High positive
0.5 to 0.7	Moderate positive
0.3 to 0.5	Low positive
-0.3 to 0.3	Little, if any
-0.5 to -0.3	Low negative
-0.7 to -0.5	Moderate negative
-0.9 to -0.7	High negative
-1.0 to -0.9	Very high negative

$$R = \frac{n\sum_{i=1}^{n} x_i y_i \; - \; n^2\,\bar{x}\bar{y}}{\sqrt{n\sum_{i=1}^{n} x_i^2 - (n\bar{x})^2}\;\sqrt{n\sum_{i=1}^{n} y_i^2 - (n\bar{y})^2}}$$



Important Note: Positive correlation means that as one variable increases, the other variable also increases. Negative correlation means that as one variable increases, the other variable decreases.

Least Squares

We can generalize this approach to any functional form.

$$S = \sum_1^n [y_i - f(x_i)]^2 \qquad (4.3\text{-}4)$$

Where $f(x; a_0, a_1, \cdots, a_m)$

Linear forms are ones for which:

$$f(x) = a_0 f_0(x) + a_1 f_1(x) + \cdots + a_m f_m(x)$$

$$f(x) = \sum_{j=0}^m a_j f_j(x) \qquad (4.3\text{-}5)$$

Where the a_j represent a set of basis functions that are typically orthogonal. This leads to a linear system for the a_j coefficients.

Polynomial basis functions have the form

$$f_j(x) = x^j \ (j = 0,1,\cdots,m) \qquad (4.3\text{-}6)$$

The derivatives, $\frac{\partial S}{\partial a_j} = 0$, lead to a linear system for the a_j coefficients.

Likewise, exponential functions can be used by employing logs, so that if

$$f(x) = ae^{bx}$$

We can recast the function as

$$F(x) = lnf(x) = lna + bx = A + bx \qquad (4.3\text{-}7)$$

And optimize the coefficients of Eq. 4.3-7.

ployFit – polynomial least squares

plotPoly – plots the curve fit and data

Other functions are possible, but don't necessarily lead to linear system, so are less often used.

Unit 4.4 – Neural Networks

The idea of Neural Networks is to create functional relationships that mimic how our brains process information by exploiting the idea of neurons connected to one another. These connections are sometimes called channels and adjusted by weights and biases.

In a mathematical sense, we develop functional relationships between variables, such as:

$$f(x) = ax^2 + bx + c \qquad (4.4\text{-}1)$$

Which could be represented as:

Where the blue box represents Eq. 4.4-1. We could extend this to multiple variables, such as:

$$f(x,y) = ax^2y + bxy^2 + c \qquad (4.4\text{-}2)$$

And then

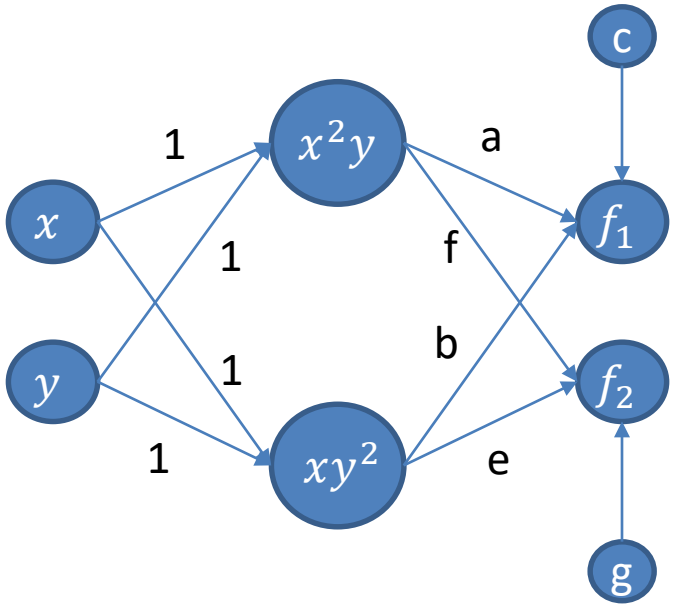
Or maybe even some vector output:

$$f_1(x,y) = ax^2y + bxy^2 + c \qquad (4.4\text{-}3)$$

$$f_2(x,y) = exy^2 + fx^2y + g \qquad (4.4\text{-}4)$$

Represented as:

Another way to look at this might be to consider the variable groups as neurons, i.e.,



Where the 1's, a,b,e,f are weights, c and g are biases, x^2y and xy^2 are activation functions, and all the circles are neurons.

You probably won't see a neural network built this way. The more common approach is summarized in the figure below:

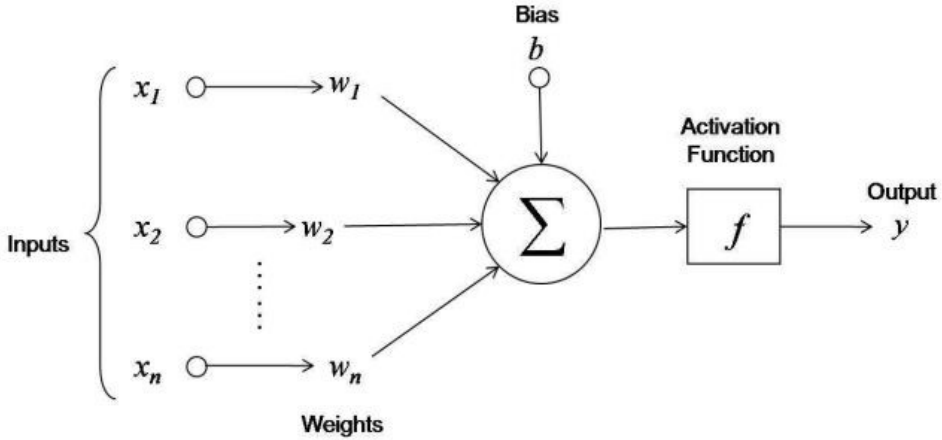


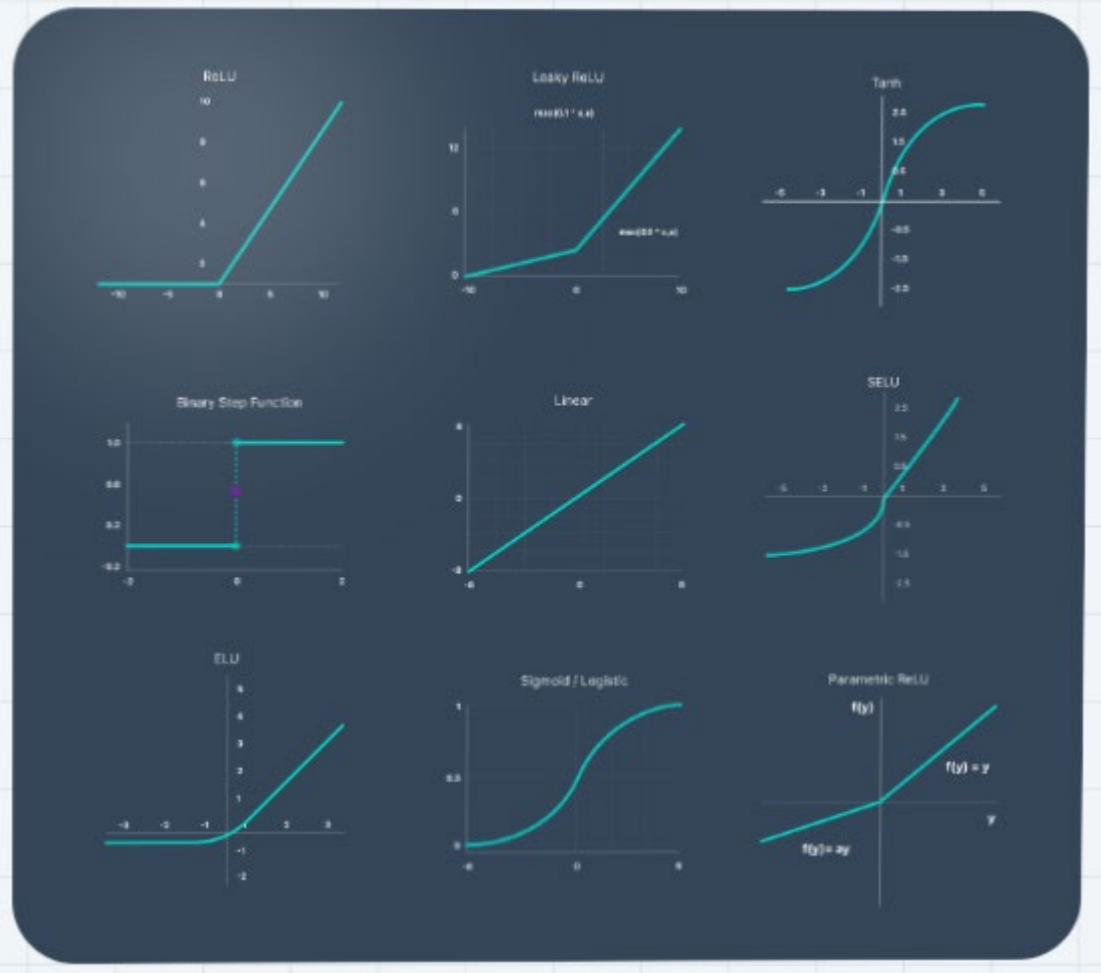
Figure 2 — Operations done by a neuron

Which comes from the recommended reading website:

https://towardsdatascience.com/first-neural-network-for-beginners-explained-with-code-4cfd37e06eaf

Go to this website AFTER watching the video on Slide 4.

We still need to define the activation function, of which many exist:



Which can be found at: https://www.v7labs.com/blog/neural-networks-activation-functions

Go to this website AFTER watching the video on Slide 4 and website on Slide 2.

The more standard picture you see of a neural network is something like this:

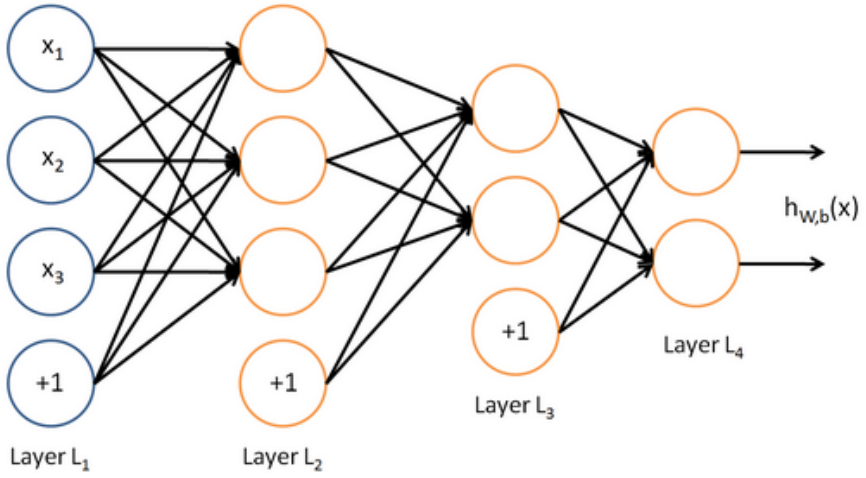


Figure 1 — Representation of a neural network

Where L_1 is the input layer, L_4 is the output layer, and L_2 and L_2 are hidden layers.

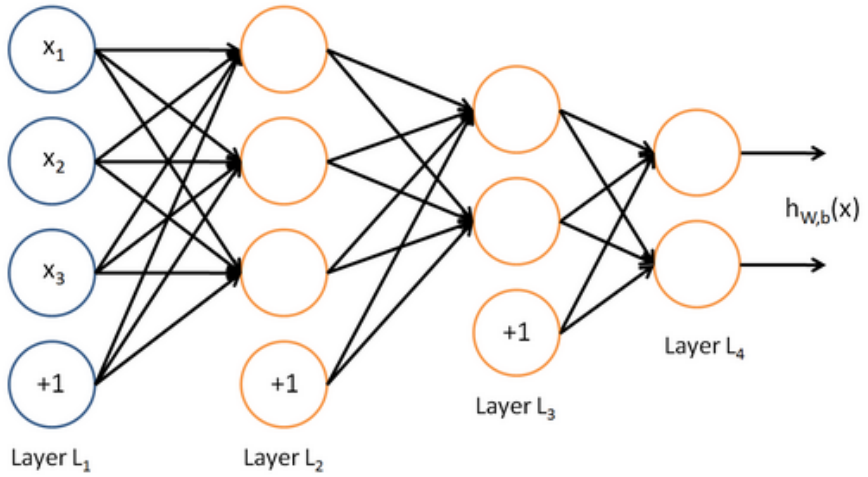


Figure 1 — Representation of a neural network

A deep neural network is one with two or more hidden layers.

With given weights and biases the outputs are defined for any given set of inputs. However, their choice depends on an idea called back propagation in which the result for a set of known target training data is utilized to increase or decrease the weights and biases. A nice discussion of this is found at:

https://www.youtube.com/watch?v=bfmFfD2RIcg

I suggest having a look at this video as your first stop outside the notes.

NN Terminology

The process of training a neural network on a full set of data is iterative. Every time that this process is invoked for all of the data is called an epoch. There are many different algorithms for performing the training, but the gradient descent method is a common approach.

The learning rate is essentially an idea that describes how quickly the weights and biases are permitted to change.

Training data – The target data that a neural network is attempting to represent.

Validation data – Data not included in the training set that is used to see how closely our trained model represents other data. Necessary to avoid overfitting.

Testing data – Also data not included in the other sets. Similar to validation data but is meant to see how the network performs on new cases.