Linear Models

In this section, we review some basics of modeling via linear algebra: finding a line of best fit, Hebbian learning, pattern classification.

Best Fitting Line

In this section, we examine the simplest case of fitting data to a function. We are given n ordered pairs of data:

$$X = \{x_1, x_2, \ldots, x_n\} \quad Y = \{y_1, y_2, \ldots, y_n\}$$

We wish to find the best linear relationship between X and Y. But what is "best"? It depends on how you look at the data, as described in the next three exercises.

1. **Exercise:** Let y be a function of x. Then we are trying to find b_0 and b_1 so that

$$y = b_0 x + b_1$$

best describes the data. If the data were perfectly linear, then this would mean that:

$$\begin{array}{rcl} y_1 &=& b_0 x_1 + b_1 \\ y_2 &=& b_0 x_2 + b_1 \\ & \vdots & \\ y_n &=& b_0 x_n + b_1 \end{array} \quad \Rightarrow \quad \mathbf{y} = \begin{bmatrix} & 1 \\ \mathbf{x} & 1 \\ & \vdots \\ & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \end{bmatrix} \quad \Rightarrow \quad A\mathbf{b} = \mathbf{y}$$

However, most of the time the data is not actually, *exactly* linear, so that the values of y don't match the line: $b_0 x + b_1$. There are many ways of expressing the error- Below, we look at three ways.

The first way is to define the error as the following:

$$E_1 = \sum_{k=1}^{n} |y_k - (b_0 x_k + b_1)|$$

(a) Show graphically what this error would represent for one of the data points.

(b) E_1 is a function of two variables, b_0 and b_1. What is the difficulty in determining the minimum[1] error using this error function?

The usual method of defining the error is to sum the squared errors up:

$$E_{\text{se}} = \sum_{k=1}^{n} (y_k - (b_0 x_k + b_1))^2$$

Why is it appropriate to use this error instead of the other error?

[1] We could solve this problem by using Linear Programming, but that is outside the scope of these notes- but maybe later!

(c) E_{se} is a function of b_0 and b_1, so the minimum value occurs where

$$\frac{\partial E_{\text{se}}}{\partial b_0} = 0 \quad \frac{\partial E_{\text{se}}}{\partial b_1} = 0$$

This leads to the system of equations: (the summation index is 1 to n)

$$\begin{aligned} b_0 \sum x_k^2 + b_1 \sum x_k &= \sum x_k y_k \\ b_0 \sum x_k + b_1 n &= \sum y_k \end{aligned}$$

(d) Show that this is the same set of equations you get by solving the normal equations, $A^T A\mathbf{b} = A^T\mathbf{y}$, assuming A is full rank.

(e) **Exercise:** Write a Matlab routine that will take a $2 \times n$ matrix of data, and output the values of $b_0 = m$ and $b_1 = b$ found above. The first line of code should be:

```
function [m,b]=Line1(X)
```

and save as `Line1.m`.

The last case is where we treat x and y independently, so that we don't assume that one is a function of the other.

(f) Show that, if $ax + by + c = 0$ is the equation of the line, then the distance from (x_1, y_1) to the line is

$$\frac{|ax_1 + by_1 + c|}{\sqrt{a^2 + b^2}}$$

which is the size of the orthogonal projection of the point to the line. This is actually problem 53, section 11.3 of Stewart's Calculus text, if you'd like more information.

(HINT: The vector $[a, b]^T$ is orthogonal to the line $ax+by+c = 0$. Take an arbitrary point P on the line, and project an appropriate vector to $[a, b]^T$.)

Conclude that the error function is:

$$E = \sum_{k=1}^{n} \frac{(ax_k + by_k + c)^2}{a^2 + b^2}$$

(g) Draw a picture of the error in this case, and compare it graphically to the error in the previous 2 exercises.

(h) The optimimum value of E occurs where $\frac{\partial E}{\partial c} = 0$. Show that if we mean subtract X and Y, then we can take $c = 0$. This leaves only two variables.

(i) Now our error function is:

$$E = \sum_{k=1}^{n} \frac{(ax_k + by_k)^2}{a^2 + b^2}$$

Show that we can transform this function (with appropriate assumptions) to:

$$E = \sum_{k=1}^{n} \frac{(x_k + \mu y_k)^2}{1 + \mu^2}$$

(for some μ), and conclude that E is a function of one variable.

(j) Now the minimum occurs where $\frac{dE}{d\mu} = 0$. Compute this quantity to get:

$$\mu^2 A + \mu B + C = 0$$

where A, B, C are expressions in $\sum x_k y_k$, $\sum x_k^2$, $\sum y_k^2$. This is a quadratic expression in μ, which we can solve. Why are there (possibly) 2 real solutions?

(k) Write a Matlab routine `[a,b,c]=Line2(X)` that will input a $2 \times n$ matrix, and output the right values of a, b, and c.

2. **Matlab Exercise:** Try the 2 different approaches on the following data set, which represents heights (in inches) and weight (in lbs.) of 10 teenage boys. (Available in `HgtWgt.mat`)

X	69	65	71	73	68	63	70	67	69	70
Y	138	127	178	185	141	122	158	135	145	162

Plot the data with the 3 lines. What do the 3 approaches predict for the weight of someone that is 72 inches tall?

3. **Exercise:** Do the same as the last exercise, but now add the data point $(62, 250)$. Compare the new lines with the old. Did things change much?

The Median-Median Line:

The median of data is sometimes preferable to the mean, especially if there exists a few data points that are far different than "most" data.

1. **Definition:** The *median* of a data set is the value so that exactly half of the data is above that point, and half is below. If you have an odd number of points, the median is the "middle" point. If you have an even number, the median is the average of the two "middle" points. Matlab uses the `median` command.

2. **Exercise:** Compute (by hand, then check with Matlab) the medians of the following data: $\{1, 3, 5, 7, 9, 11\}$

The motivation for the median-median line is to have a procedure for line fitting that is not as sensitive to "outliers" as the 3 methods in the previous section.

Median-Median Line Algorithm

- Separate the data into 3 equal groups (or as equal as possible). Use the $x-$axis to sort the data.
- Compute the median of each group (first, middle, last).
- Compute the equation of the line through the first and last median points.
- Find the vertical distance between the middle median point and the line.
- Slide the line 1/3 of the distance to the middle median point.

3. **Exercise:** The hardest part of the Matlab code will be to partition the data into three parts. Here are some hints:

   - If we divide a number by three, we have three possible remainders: $0, 1, 2$. What is the most natural way of seperating data in these three cases (i.e., if we had 27, 28 or 29 data points)?
   - Look at the Matlab command `rem`. Notice that:

     ```
     rem(27,3)=0   rem(28,3)=1   rem(29,3)=2
     ```

   - The command to sort: `[s,index]=sort(x)`. For example,

     ```
     >> a=[3 2 4 1];
     >> [a2,idx]=sort(a)
     a2 =
          1     2     3     4
     idx =
          4     2     1     3
     ```

     Notice that $a_2(j) = a(idx(j))$. We can therefore sort x first, then sort y according to the index for x.

4. **Exercise:** Try this algorithm on the last data set, then add the new data point. Did your lines change as much?

5. **Exercise:** Consider the following data set [1] which relates the index of exposure to radioactive contamination from Hanford to the number of cancer deaths per $100,000$ residents. We would like to get a relationship between these data. Use the four techniques above, and compare your answers. Compute the actual errors for the first three types of fits and compare the numbers.

County/City	Index	Deaths
Umatilla	2.5	147
Morrow	2.6	130
Gilliam	3.4	130
Sherman	1.3	114
Wasco	1.6	138
Hood River	3.8	162
Portland	11.6	208
Columbia	6.4	178
Clatsop	8.3	210

Hebbian Learning

D.O. Hebb (1904-1985) was a physiological psychologist at McGill University. We will discuss his influence on machine learning, but he also did some very interesting experiments to determine how the brain functions. Among the research[2]:

[2] As usual, these footnote sketches are somewhat oversimplified. Read about Hebb's work from his writings, especially "The Organization of Behavior" (1949), and his autobiography in 1980 "A History of Psychology through Autobiography", vol VII

- Evidence for the plasticity of the brain:

  > Much to Hebb's amazement, he found that even after substantial loss of tissue from the frontal lobes of the brain, there was no loss in intelligence, and in some cases, he even detected a gain in intelligence[3].

- Sensory Deprivation experiments: Sensory input is vital for brain functioning. Over time without sensory stimulation, subjects hallucinate and their personalities begin to break down.

In Hebb's view, learning could be described physiologically. That is, there is a physical change in the nervous system to accommodate learning, and that change is summarized by what we now call Hebb's postulate (from his 1949 book):

> When an axon of cell A is near enough to excite a cell B and repeatedly takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased.

As with many named theorems and postulates, this was not an idea that was completely new, but he does give the postulate in a form that can be used as a basis for machine learning.

Next, we look at a mathematical model of Hebb's postulate.

Linear Neurons and Hebbian Learning

Let us first build a simple model for a neuron. A neuron has three basic parts- The dendrites, which carry information to the cell body, the cell body, and the axon, which carries information away from the cell body.

Multiple signals come in to the cell body from the dendrites. Mathematically, we will assume they all arrive at the same time, and the action of the dendrites (or the arrival site of the cell body) is that each signal is changed by the physiology of the cell. That is, if x_i is "information" along dendrite i, arrival at the cell body changes it to $w_i x_i$, where w_i is some real scalar. For example, $w_i > 1$ is an amplification of the signal, $0 < w_i < 1$ is an inhibition of the signal, and negative values mean re-polarization.

Next, the cell body collates this information by summing these signals together. This action is easily represented by the inner product of the vector of w's (the *weights*) to the signal x. For the purposes of this section, we will assume no further processing. Thus, for one neuron with n incoming signals, the input-output relationship is:

$$\mathbf{x} \mapsto \mathbf{w} \cdot \mathbf{x}$$

If the signal is passed to a *cell assembly*, or group of neurons, then each neuron has its own set of weights, and the mapping becomes:

$$\mathbf{x} \to W\mathbf{x} = \mathbf{y}$$

If we have k neurons and $\mathbf{x}$ is a vector in $\mathbb{R}^n$, then W is a $k \times n$ matrix, and each row corresponds to a signal neuron's weights (that is, W_{ij} refers to the weight taking x_j to neuron i).

[3] An Introduction to the Theories of Learning, p. 394

Next, we model Hebb's postulate. We suppose that we present the network with a pattern, $\mathbf{x} \in \mathbb{R}^n$, and it outputs a pattern, $\mathbf{y} \in \mathbb{R}^k$. In terms of each weight, we see that

W_{ij} connects the j^{th} value of the input to the i^{th} value of the output.

Thus we might take the following as Hebb's Rule. The change in the weight connecting the j^{th} input to the i^{th} cell is given by:

$$\Delta W_{ij} = \alpha y_i x_j$$

where α is called **the learning rate**. If both x_j and y_i match in sign, then W_{ij} becomes larger. If there is a mismatch in sign, W_{ij} gets smaller. This is the *unsupervised Hebbian rule.* We now should formulate this using matrix algebra.

As before, assume we have n inputs to the network, and m outputs. Then W is $m \times n$, with $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{y} \in \mathbb{R}^m$. If we compute the outer product, $\mathbf{y}\mathbf{x}^T$, we get:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix} [x_1, x_2, \ldots, x_n] = \begin{bmatrix} y_1x_1 & y_1x_2 & \ldots & y_1x_n \\ y_2x_1 & y_2x_2 & \ldots & y_2x_n \\ \vdots & & & \vdots \\ y_mx_1 & y_mx_2 & \ldots & y_mx_n \end{bmatrix}$$

You should verify that in this case, we can compactly write Hebb's rule as:

$$W_{\text{new}} = W + \alpha \mathbf{y}\mathbf{x}^T$$

and that this change is valid for a single $\mathbf{x}, \mathbf{y}$ stimulus-response pair. If we define W_0 to be the initial weight matrix (we could initialize it randomly, for example), then the update rule becomes:

$$W_1 = W_0 + \alpha \mathbf{y}\mathbf{x}^T = W_0 + \alpha W_0 \mathbf{x}\mathbf{x}^T = W_0 \left(I_{n\times n} + \alpha \mathbf{x}\mathbf{x}^T \right)$$

EXERCISES:

1. Write W_n in terms of W_0 using the previous formula as a starting point.
2. Show that, if λ_i, $\mathbf{v}_i$ is the eigenvalue and eigenvector of a matrix A, then $(1 + \beta\lambda_i)$ and $\mathbf{v}_i$ is an eigenvalue and eigenvector of $(I + \beta A)$.
3. Let the matrix $A = \mathbf{x}\mathbf{x}^T$, where $\mathbf{x} \in \mathbb{R}^n$, and $\mathbf{x} \neq 0$. If $\mathbf{v} \in \mathbb{R}^n$, show that $A\mathbf{v}$ is a scalar multiple of x. (This shows that the dimension of the columnspace of $\mathbf{x}\mathbf{x}^T$ is 1)
4. Same matrix A as in the previous exercise. Show that one eigenvalue is $\|x\|^2$ (Hint: The eigenvector is $\mathbf{x}$).
5. We'll state the following without proof for now: If A is $n \times n$ and symmetric, and the columnspace of A has dimension 1, then there is exactly one nonzero eigenvalue. Use this, together with the previous exercises to compute the eigenvalues of $I + \alpha\mathbf{x}\mathbf{x}^T$.
6. There is a theorem that says that if the eigenvalues satisfy $|\lambda_i| \leq 1$, then the elements of A^n will converge (otherwise, the elements of A^n will diverge).

   Given our previous computation, will Hebb's rule give convergence?

Hebb's Rule with Feedback

Somehow, we want to take feedback into account so that we can use Hebb's rule in supervised learning problems.

Let $\mathbf{t}$ be the target (or desired) value for the input $\mathbf{x}$. That is, we would be given pairs of vectors,

$$(\mathbf{x}_i, \mathbf{t}_i)$$

and we want to build an affine function using matrix W and vector $\mathbf{b}$ so that

$$W\mathbf{x}_i + \mathbf{b} = \mathbf{t}_i$$

In the supervised Hebbian rule, we need to update the weights based on what we want the network to do, rather than what the network is already doing:

$$\Delta W_{ij} = \alpha t_j x_i$$

There is still something unsatisfying here- When should we stop the training? It seems like the weights could diverge as training progresses, and furthermore, we're not taking the actual outputs of the network into account. Heuristically, we would like for the update to go to zero as the target values approach the network outputs. This leads us to our final modification of our basic Hebb rule, and is called the Widrow-Hoff learning rule[4]:

$$\Delta W_{ij} = \alpha(t_j - y_j)x_i$$

If we put this in matrix form, the learning rule becomes:

$$W^{\text{new}} = W^{\text{old}} + \alpha\left(\mathbf{t} - \mathbf{y}\right)\mathbf{x}^T$$

where $(\mathbf{x}, \mathbf{t})$ is a desired input-output relation, and $\mathbf{y} = W\mathbf{x}$.

Additionally, sometimes it is appropriate to add a bias term so that the network has the output:

$$\mathbf{y} = W\mathbf{x} + \mathbf{b}$$

The bias update is similar to the previous update:

$$\mathbf{b}^{\text{new}} = \mathbf{b}^{\text{old}} + \alpha\left(\mathbf{t} - \mathbf{y}\right)$$

Example: Associative Memory

Here we will reproduce an experiment by Widrow and Hoff[5] who built an actual machine to do this (we'll do a computer simulation).

We'll have three letters as input, T, G and F. We'll associate these letters to the numbers $-60, 0, 60$ respectively. We want our network to perform the association using the Widrow-Hoff learning rule.

The letters will be defined by 4×4 arrays of numbers, where 1 corresponds to the color black, and -1 corresponds to the color white. In this example, we'll have two samples of each letter, as shown in Figure 1.

Implementation:

[4] Also goes by the names Least Mean Squares rule, and the delta rule.

[5] See "Adaptive Switching Circuits" by B. Widrow and M.E. Hoff, in 1960 IRE WESCON Convention Record, New York: IRE, Part 4, p. 96-104. You might find reprints also on the internet.

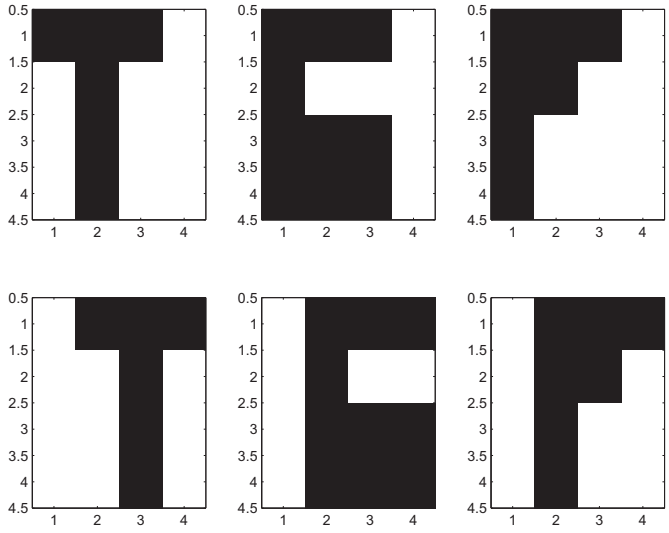

Figure 1: The inputs to our linear associative memory model: Three letters, T, G, H, where we have two samples of each letter, and each letter is defined by a 4×4 grid of numbers. We'll be associating T with -60, G with 0, and H with 60.

- First, we process the input data. Rather than working with 4×4 grids, we concatenate the columns to work with vectors in $\mathbb{R}^{16}$. Thus, we have 6 domain data points in $\mathbb{R}^{16}$, two samples of each letter. Construct range points so that they correspond with the letters.
- We'll use an $\alpha = 0.03$.
- We'll take several passes through all the data points.
- To measure the error, after each pass through the data, we'll put each letter through the function to get an output value. We'll take the square of the difference between that and the desired value. We'll take the sum of the errors squared for those six samples to be the measure of the error for that pass.

Here is the code we used for this example. Again, be sure to read and understand what the code is doing. A lot of the initial part of the code is just there to get the data read in and plotted.

```
T1=[1 1 1 -1
    -1 1 -1 -1
    -1 1 -1 -1
    -1 1 -1 -1];
T2=[-1 1 1 1
    -1 -1 1 -1
    -1 -1 1 -1
    -1 -1 1 -1];
G1=[1 1 1 -1
    1 -1 -1 -1
    1 1 1 -1
```

```
    1 1 1 -1];
G2=[-1 1 1 1
    -1 1 -1 -1
    -1 1 1 1
    -1 1 1 1];
F1=[1 1 1 -1
    1 1 -1 -1
    1 -1 -1 -1
    1 -1 -1 -1];
F2=[-1 1 1 1
    -1 1 1 -1
    -1 1 -1 -1
    -1 1 -1 -1];

gg=colormap(gray);
gg=gg(end:-1:1,:);

subplot(2,3,1)
imagesc(T1)
colormap(gg)
subplot(2,3,2)
imagesc(G1)
subplot(2,3,3)
imagesc(F1)

subplot(2,3,4)
imagesc(T2)
subplot(2,3,5)
imagesc(G2)
subplot(2,3,6)
imagesc(F2)

%************************************
%The main code starts here!
%************************************

X=[T1(:) T2(:) G1(:) G2(:) F1(:) F2(:)];
T=[60 60 0 0 -60 -60];
alpha=0.03;
W=randn(1,16);
b=0;
TotalErr=0;
NumPoints=6;

for k=1:60
    idx=randperm(6);
```

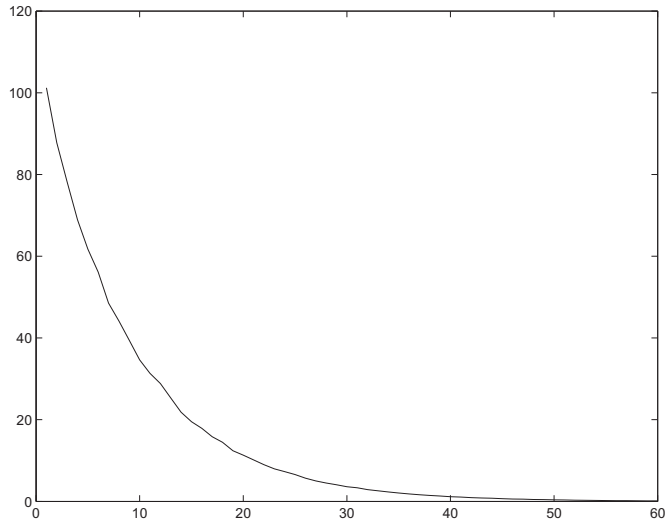

Figure 2: The errors for the Widrow-Hoff rule applied to letter recognition (or associative memory. After 60 passes through the data, the associations are very good.

```
    for j=1:NumPoints
        ThisOut=W*X(:,idx(j))+b;
        ThisErr=T(idx(j))-ThisOut;
    %Update the weights and biases
        W=W+alpha*ThisErr*X(:,idx(j))';
        b=b+alpha*ThisErr;
    end
    EpochErr(k)=norm((W*X+b*ones(1,6))-T);
end
figure(2)
plot(EpochErr);
```

The plot of the error is shown in Figure 2. The horizontal axis counts the number of passes through the data, and the vertical axis gives the sum of the squared errors. Note that after 60 passes, we get very good classification of the letters!

You should put this code into Matlab and reproduce the figures.

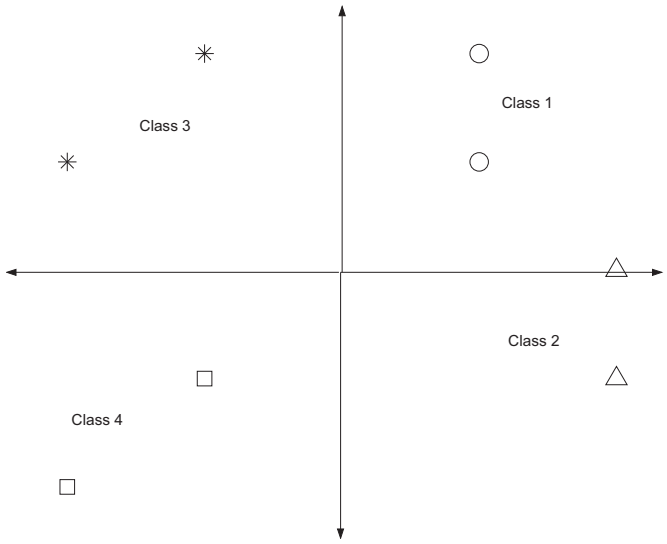

Figure 3: Pattern Classification Problem. Each point is a sample of one of the four classes.

Pattern Classification

Let's put all of this together to solve a pattern classification problem. Suppose we are given the following associations:

Point	Class
$(1, 1)$	1
$(1, 2)$	1
$(2, -1)$	2
$(2, 0)$	2
$(-1, 2)$	3
$(-2, 1)$	3
$(-1, -1)$	4
$(-2, -2)$	4

Graphically, we can see the classes in the plane in Figure 3. There are several ways of performing the desired mapping- for example, the outputs could be $1, 2, 3, 4$. But this may have unintended consequences. In this case, the metric between outputs would imply that Class 4 is much farther away from Class 1 than Class 3. A better method may be to take Class 1 to be the vector $[-1, -1]^T$, Class 2 is the vector $[-1, 1]^T$, Class 3 is $[1, -1]^T$, and Class 4 is $[1, 1]^T$. Now the 4 classes are on the vertices of a square.

Now for the details of the program. First write the inputs as an 2×8 matrix, with a corresponding output matrix that is also 2×8. Parameters that can be placed first will be the maximum number of times through the data N and the learning rate, a, which we will set to 0.04. We can also set an error bound so that we might stop early. Set the initial weights to the 2×2 identity, and the bias vector b to $[1, 1]^T$.

Check your algorithm using the following data: After using the first data point, W and b should have the values:

$$W = \begin{bmatrix} 0.88 & -0.12 \\ -0.12 & 0.88 \end{bmatrix}, \quad b = \begin{bmatrix} 0.88 \\ 0.88 \end{bmatrix}$$

Let the program run until you think it has converged (this is your choice). Then, plot the points $\mathbf{x} = [x, y]^T$ so that $W\mathbf{x} + \mathbf{b} = \mathbf{0}$, which will be the two lines:

$$W_{11}x + W_{12}y + b_1 = 0, \quad W_{21}x + W_{22}y + b_2 = 0$$

These lines form what is called the *decision boundary.*

Linear Networks, continued

Theorem: A multilayer linear neural network is equivalent to a single layer linear neural network.

Proof: Suppose that the network has "n" nodes in the input layer, and has $N_1, N_2, \ldots, N_k$ nodes in the k hidden layers, with m nodes in the output layer. Then the mapping from the input layer to the first layer is an affine mapping from $\mathbb{R}^n$ to $\mathbb{R}^{N_1}$:

$$\boldsymbol{x} \mapsto A_1\boldsymbol{x} + \boldsymbol{b}_1$$

where A is $N_1 \times n$. Going to the next layer is realized as an affine mapping from $\mathbb{R}^{N_1}$ to $\mathbb{R}^{N_2}$

$$A_2(A_1\boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2 \doteq \hat{A}_1\boldsymbol{x} + \hat{\boldsymbol{b}}_1$$

Continuing to the end, we get one affine mapping from $\mathbb{R}^n$ to $\mathbb{R}^m$:

$$\hat{A}\boldsymbol{x} + \hat{\boldsymbol{b}}$$

where $\hat{A}$ is $m \times n$, and $\hat{\boldsymbol{b}}$ is $m \times 1$.

We have algorithms for solving systems of the form $A\mathbf{x} = \mathbf{y}$- Can we use these to solve the affine problem

$$A\boldsymbol{x} + \boldsymbol{b} = \boldsymbol{y}$$

Consider the following example

$$\begin{pmatrix} a_1 & a_2 \\ a_3 & a_4 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$

This is equivalent to the linear problem:

$$\begin{pmatrix} a_1 & a_2 & b_1 \\ a_3 & a_4 & b_2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ 1 \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$$

If we only want a two dimensional output, we can leave off the last row (the last row is there to do "perspective projections").

Exercise: Show that the system of affine equations: $AX + B = Y$ is also equivalent to a linear problem: $\hat{A}\hat{X} = Y$