What are neural networks?

(requires only high-school maths)

甄景贤 (King-Yin Yan)

General.Intelligence@Gmail.com

The 3 main approaches in artificial intelligence are:

- logic
- neural networks
- evolution

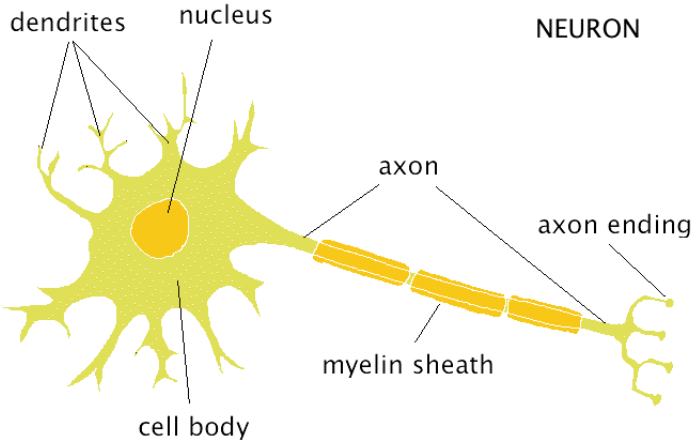
Neural networks is a special kind of **statistical learning**, that operates on "points" in a vector space.

Deep learning is the hottest technique in current AI research. "Deep" simply means "many layers of neural networks".

Biological neurons

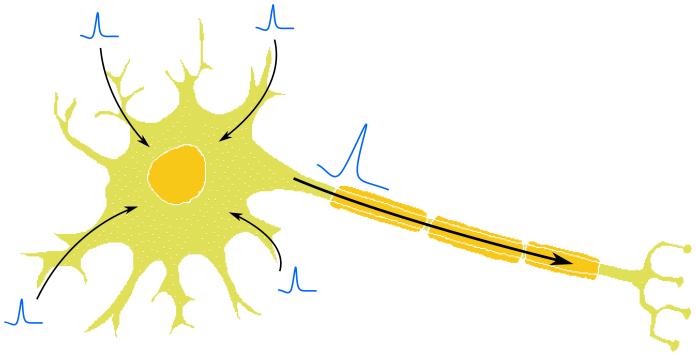
Let's refresh some high-school biology ☺

This is a biological neuron:



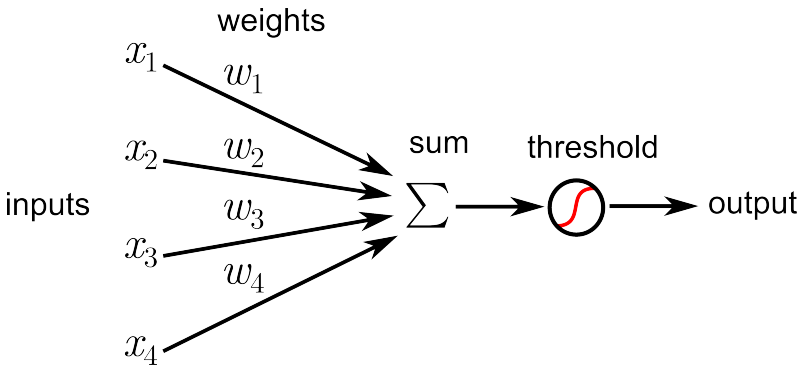
(1)

Dendrites **collect** electrical signals; When the **total** of these signals exceed a certain **threshold**, the neuron **fires** an electric impulse, sending to another neuron via the **axon**:



(2)

Mathematically, this can be greatly simplified to such a **model**:



(3)

That means: each input value is **weighted** and then summed together, and then passes through a $\int$ function for output.

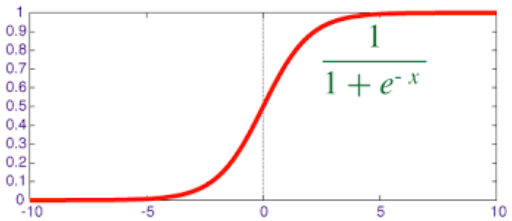
In formula:

$$\boxed{\text{output}} \quad y = \int \left[\sum_i (w_i \, x_i) \right] \tag{4}$$

where $\int$ = sigmoid function, is defined by:

$$\int(x) = \frac{1}{1 + e^{-x}} \tag{5}$$

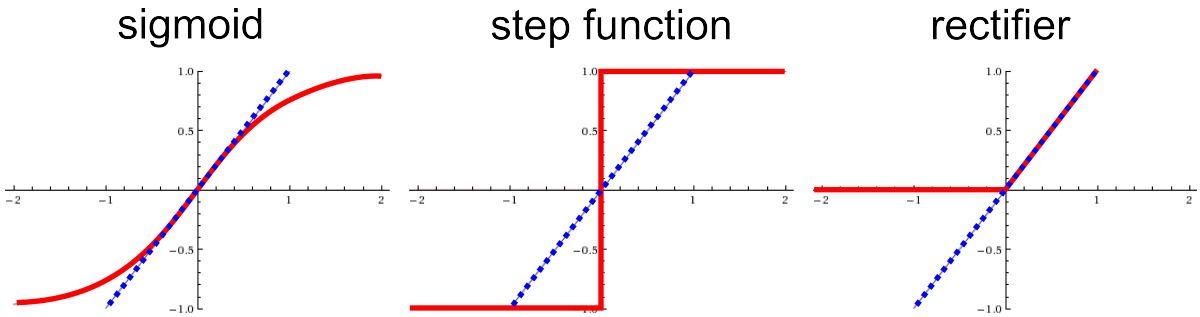
Its shape is like this:



(6)

On the left it is 0 = nothing (no signals), on the right it is 1 = "yes".

These functions are also feasible candidates for the "threshold":



(7)

Fun fact #1

The surface of the neuron is covered with sodium-potassium (Na^+/K^+) channels, that use ATP (adenosine triphosphate, the energy source in the cell) to "pump" ions into the cell with a 3 Na^+ : 2 K^+ ratio, creating a voltage difference. When the voltage exceeds the threshold, certain ion channels open, the built-up voltage is released to create an "action potential". This phenomenon can be described using differential equations, that is the famous **Hodgkin-Huxley** equation, and its simplified version, the **FitzHugh-Nagumo** equation.

A very special feature about the action potential is that it is "**all-or-nothing**", ie, if the input is below threshold, the output signal would be flat (zero). Why is it like this? That is because the human brain evolved from primitive **multi-cellular** organisms (like the jellyfish) whose cells gradually learned to use electric signals to communicate. They operated in an environment like a pool of water, in which there is a lot of **noise**. Even now the human brain is like a pool of liquid, and the body is in constant motion, resulting in **heat noise**. In order to operate within such noise, there must be a mechanism to **suppress** the noise; This is the reason for all-or-nothing. That is to say, human consciousness has **finite** information content, similar to a digital computer, and is not mysterious.

Fun fact #2

The neuron's **cell membrane** is a **lipid bi-layer**, made up of fats and cholesterol. The function of cholesterol is to make the membrane structurally stable, therefore every cell needs cholesterol. The "wires" in the brain are all made of cell membranes, so the brain is basically made up of fats and cholesterol. In particular, the pig's brain which is a kind of Chinese food, has the highest cholesterol content of all foods, many times more than eggs!

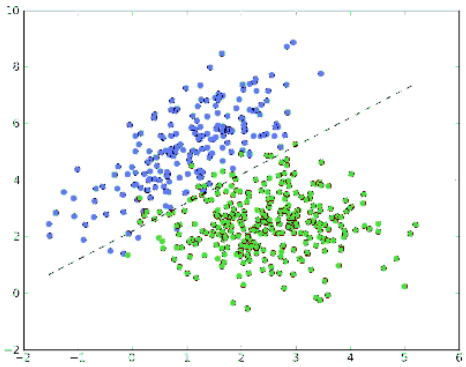
The **myelin sheath**, unique to vertebrates, is like the plastic insulation of electric wires, whose effect is to speed up electrical transmission. The octopus is an invertebrate, that's why it needs a big head to achieve the same level intelligence as vertebrate animals of comparable brain size.

Fun fact #3

When the nerve signal reaches the **synapse**, it no longer uses electrical transmission, but switches to chemical transmission with **neuro-transmitter** molecules. There are many types of neuro-transmitters, such as **serotonin** and **dopamine**, often mentioned in anti-depression drugs. But the most common neuro-transmitter is **glutamate**, the main signaling molecule for the nervous systems of all animals. Plants lack a nervous system, therefore glutamate is not found in plants. Humans like to eat meat, so we evolved a taste for meat, especially the taste for glutamate. A Japanese scientist discovered a substance in sea-weed, which when added to food mimics the taste of meat. Actually this substance is just glutamate, or MSG (mono-sodium glutamate). So MSG is not harmful to humans, it's just that we may not get a balanced nutrition if we eat MSG often instead of real meat.

1 neuron – geometric interpretation

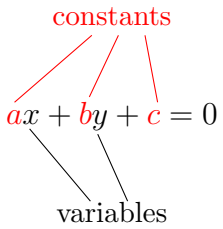
As I explained in other tutorials the goal of **machine learning** is usually to **classify** certain "points" in a space:



(8)

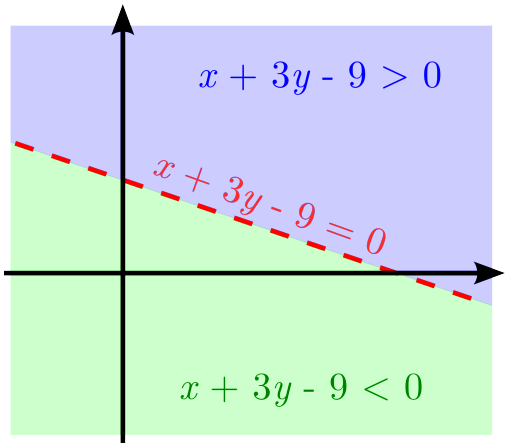
For example, in **machine vision**, a picture can have millions of **pixels**, each pixel being a dimension, its **color** is the **coordinate value** on this dimension. The entire space is the space of **all images**, with each **point** representing an image. The dimensionality of such spaces is very high (the dimension is the number of pixels per image). We often use 2 or 3 dimensions for explaining things, but the reader should use their imagination for higher dimensions.

We know from high-school maths, the equation for a **straight line** is:



(9)

Its **geometric interpretation** is like this:



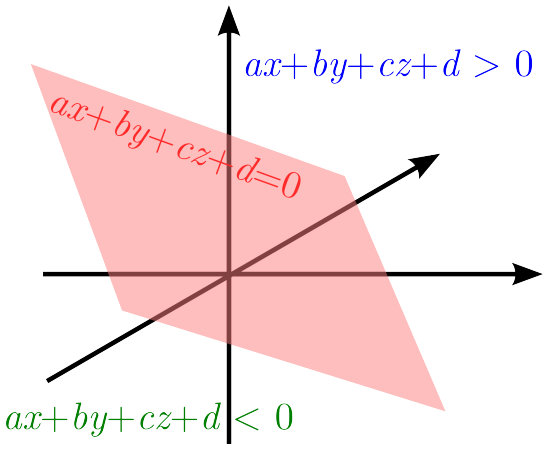
(10)

For points on the line, the equation is $= 0$. The line **cuts** the space into 2 halves: one side is > 0, the other < 0.

Generalizing to 3-dimensions, we have the equation for a **plane**:

$$ax + by + cz + d = 0 \tag{11}$$

It also **cuts** the space into 2 halves, one side > 0, the other < 0:



(12)

For arbitrary n-dimensions, with each point denoted as $\boldsymbol{x} = (x_1, x_2, ..., x_n)$, a **hyperplane** cuts the space into 2 halves, its equation is:

$$a_1 x_1 + a_2 x_2 + ... + a_n x_n + a_0 = 0 \tag{13}$$

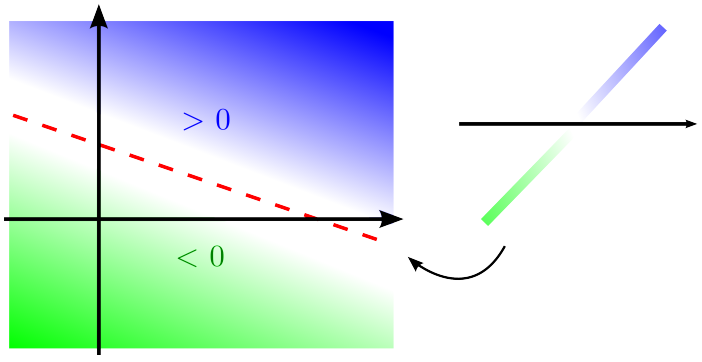
Note: What is the dimension of a hyper-plane? In 2-D space, it is a line (1-D), in 3-D space, it is a plane (2-D); In general, in n-D space a hyper-plane is an $(n-1)$-dimension object, $(n-1)$ is also called **co-dimension 1**, meaning that the ambient space is of dimension n, and equation (13) reduces the **degrees of freedom** by 1, so the object **constrained** by this equation has $n-1$ degrees of freedom.

Now we can see some resemblance between the hyper-plane and the neuron, as the neuron is a **linear combination** before passing to the $\int$:

$$\boxed{\text{output}}\ y = \int \Big[\overbrace{\sum_i (w_i\ x_i)}^{\text{linear combination}} \Big] \tag{14}$$

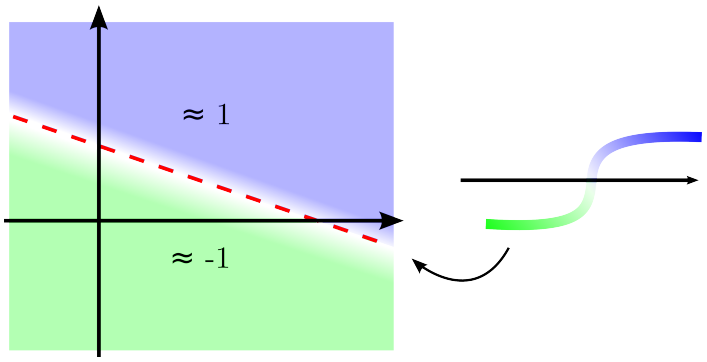
That is to say: Each neuron forms a hyper-plane, that cuts the space into 2 halves.

What if $\int$ is applied? Without $\int$, the 2 halves are > 0 and < 0; Now if we see colors as "intensity", the intensity changes gradually: (the figure on the right shows the side view, as in 3-D)



(15)

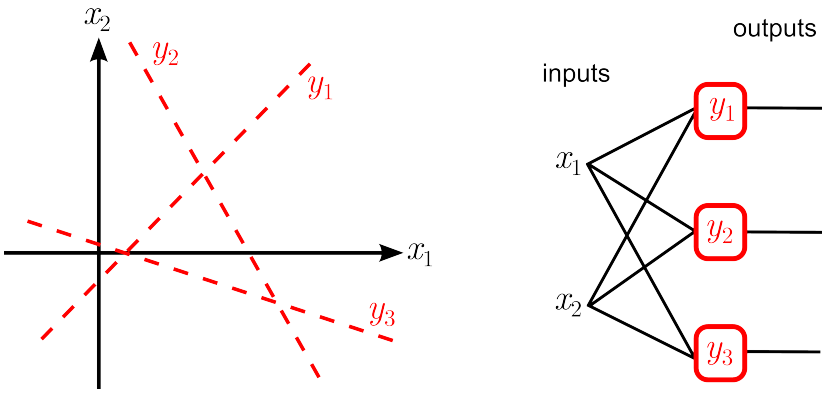
When $\int$ is applied, 1 represents "yes" and 0 represents "no", so the contrast between the 2 sides is enhanced, ie, more **polarized**, or binary:



(16)

> 1 neurons

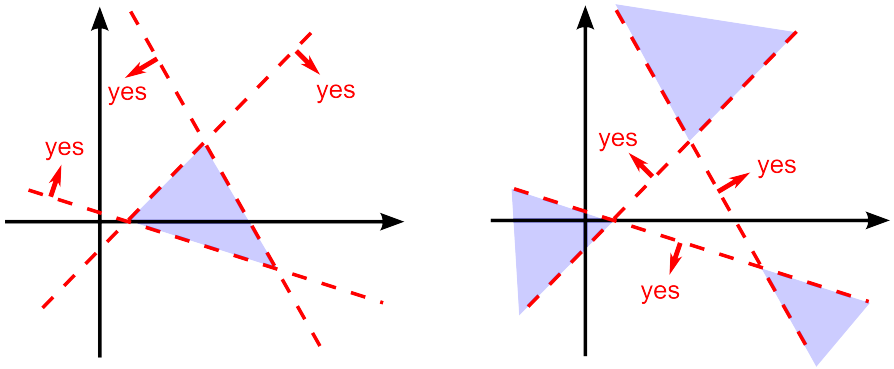
If there are > 1 neurons (on the same layer), eg. 3:



(17)

Note: The coordinates are (x_1, x_2) = input. Output is (y_1, y_2, y_3), represented by 3 dotted lines. The **network topology** of this **1 layer** of neural network is as shown on the right (The $\sum$ and $\int$ for each neuron are not shown).

Each neuron may choose one side to be "yes". The **conjunctions** of these choices may form various shapes, eg:



(18)

Also notice that in the right figure, a number of disjoint regions (separated in space) are possible.

Obviously, we could use neurons to separate and classify points in space, thus achieving the goal of **machine learning**.

1 layer of neurons

In this part we need more **linear algebra**, which I will explain in another tutorial.

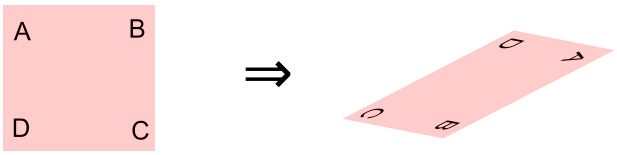
The mathematical form of 1 layer of neural network is:

$$\boxed{\text{output}}\ \boldsymbol{y} = \mathcal{S}[W\boldsymbol{x}] \tag{19}$$

where W is a matrix, ie. a **linear transformation**; $\mathcal{S}$ is a **non-linear transformation**.

Matrices are a compact way of writing **systems of linear equations**. 1 neuron $= \mathcal{S} \circ \boxed{\text{linear combination}}$, each linear combination is a **row** in a matrix.

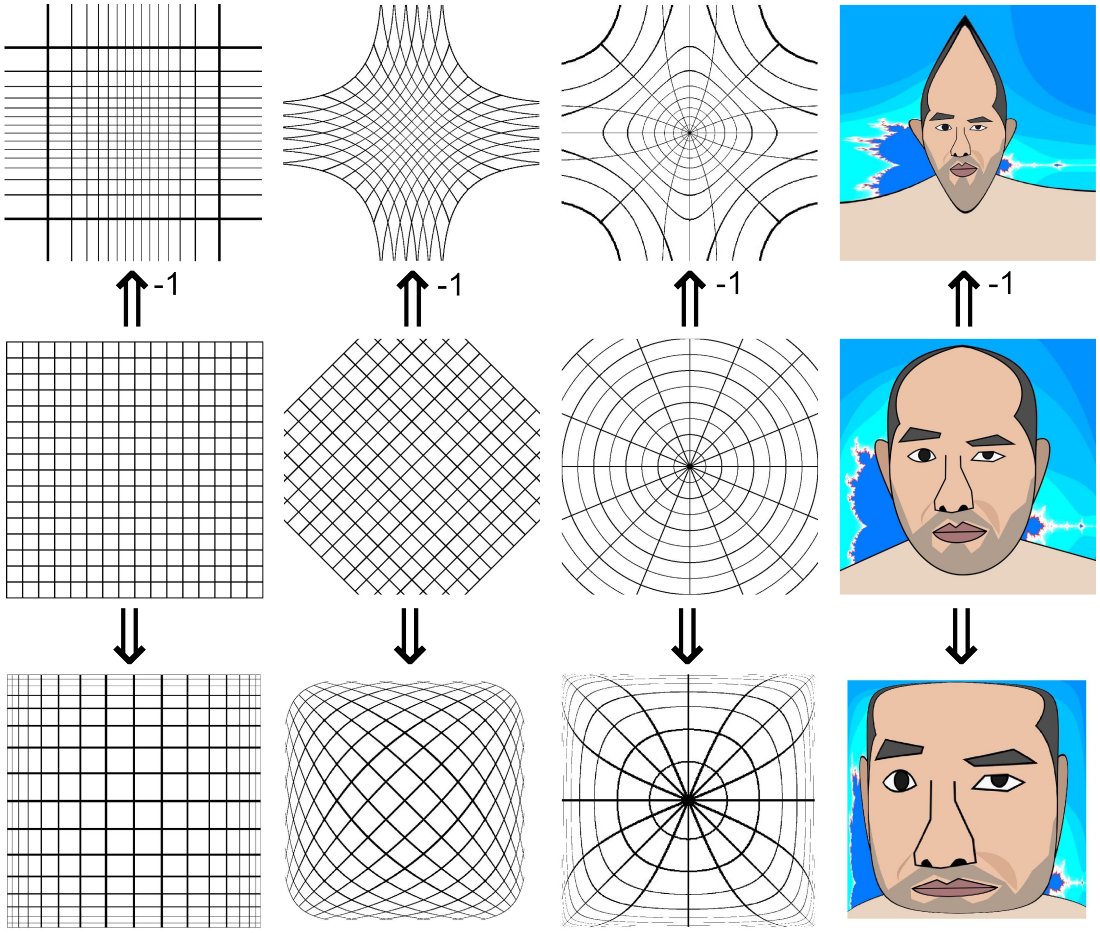
Every matrix is a representation for a **linear transformation**:



(20)

Linear transformations can "skew" a square into a parallelogram, and also include **rotations** and **translations**. Straight lines are preserved as straight lines, hence its name.

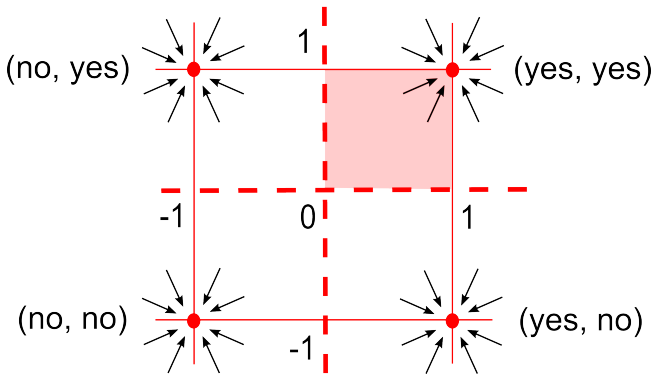
Next we need to know what $\mathcal{S}$ does: (below is the sigmoid transform of both x and y coordinates)



(21)

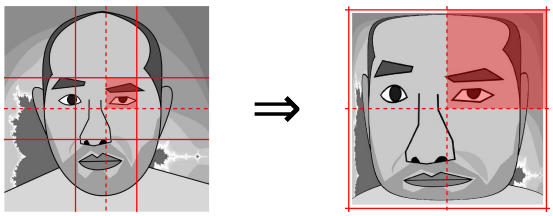
$\mathcal{S}$ can be seen as "stretching" the square to its 4 corners, so I take on the "square-face" look.

We say that these vertices are **attractors**:



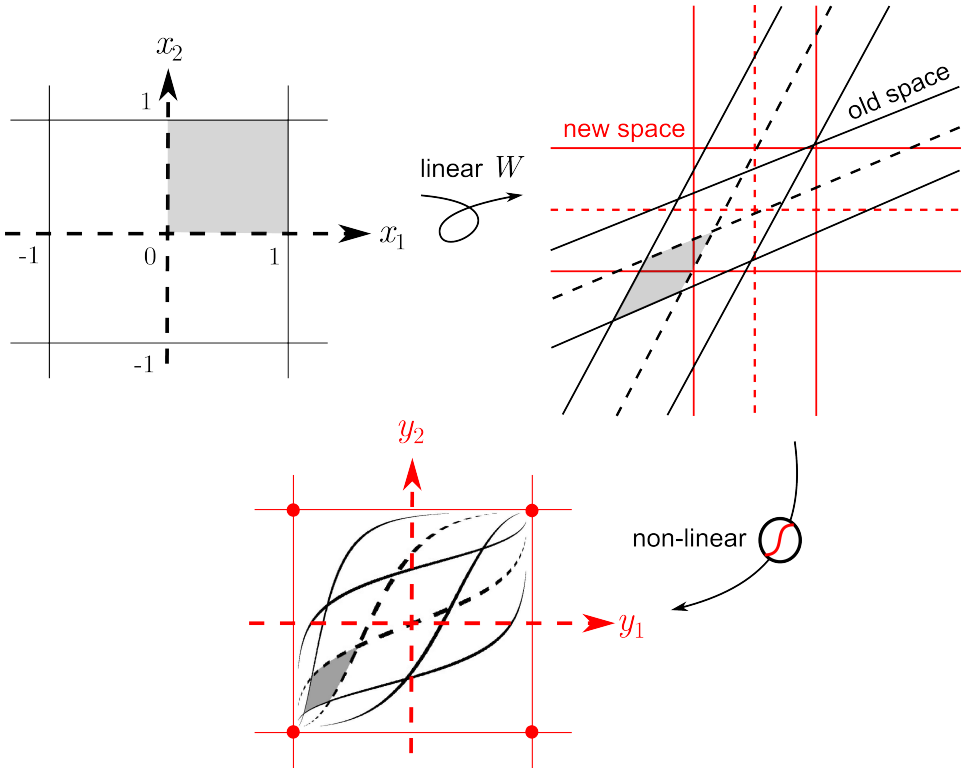
(22)

Note: During the $\int$ transform, the image is compressed from the **infinite** unbounded space to the $[-1,+1]^2$ square:



(23)

Here is the transformation performed by **1 layer** of neural network, decomposed into the W and $\int$ parts:



(24)

Code for sigmoid distortion

This Python program is adapted from Prateek Joshi's ⟨*OpenCV with Python by example*⟩, section on "image warping". It is just a simple demo, which does not use advanced image sampling techniques.
`https://www.packtpub.com/mapt/book/application_development/9781785283932`

```
import cv2
import numpy as np
import math

img = cv2.imread("input.jpg", cv2.IMREAD_GRAYSCALE)
Y, X = img.shape                    # size of the image
print "X = ", X
print "Y = ", Y

img_out = np.zeros(img.shape, dtype=img.dtype)

k = 80.0                            # scaling factor (smaller = more distorted)

for y in range(Y):
    outsideX = False                # out-of-boundary
    y0 = Y / (y + 0.5)              # y-position in original image scaled to [0,1]
    y1 = -k * math.log(y0 - 1.0)    # new position
    y2 = int(y1 + Y/2.0)            # adjust for mid-point
    if y2 >= Y or y2 < 0:           # out of boundary?
        outsideX = True
    for x in range(X):
        outsideY = False
        x0 = X / (x + 0.5)
        x1 = -k * math.log(x0 - 1.0)
        x2 = int(x1 + X/2.0)
        if x2 >= X or x2 < 0:
            outsideY = True
        if not (outsideX or outsideY):
            img_out[y,x] = img[y2,x2]
        else:
            img_out[y,x] = 255      # white color if out-of-bounds

cv2.imshow("Input", img)
cv2.imshow("Distorted", img_out)
cv2.imwrite("output.jpg", img_out)
```

The interested reader can try to make a game, where the player manually "rotate" the W matrix of a neural network, to recognize some visual patterns (perhaps projected from 3D or 4D to 2D).

The inverse of 1 layer

Now we consider the inverse of the 1-layer network: if we have a **straight line** (or "linear discriminant") in the target space, what would this line look like in the original space? This is the so-called "decision boundary", that is to say, if we make a "linear cut" in the target space, what would be the shape of our "knife" in the original space?

A linear cut in the **target** space is:

$$ax' + by' + c = 0 \tag{25}$$

The 1-layer neural network transforms it via:

$$(x'\ y') = \sigma W \begin{pmatrix} x \\ y \end{pmatrix} \tag{26}$$

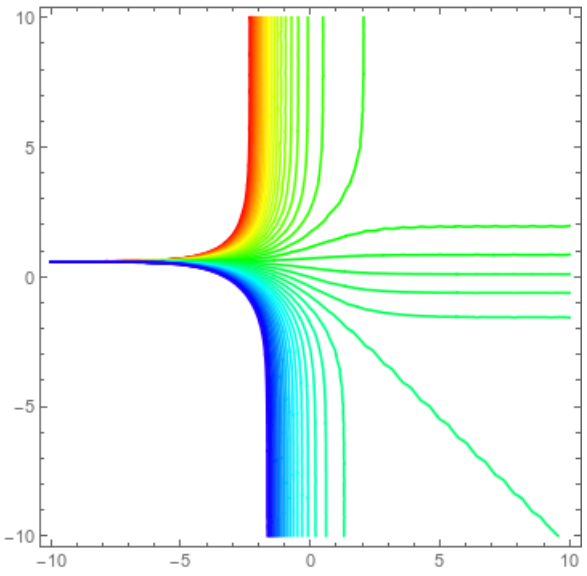
Suppose $W = \mathrm{id}$ (ignoring the effect of W), then the transform becomes:

$$\begin{cases} x' = \sigma x \\ y' = \sigma y \end{cases} \tag{27}$$

The equation of the "cut" in the original space is thus:

$$\frac{a}{1+e^{-x}} + \frac{b}{1+e^{-y}} + c = 0 \tag{28}$$

This family of curves (ie, the "shapes of the knifes") are typically like these:



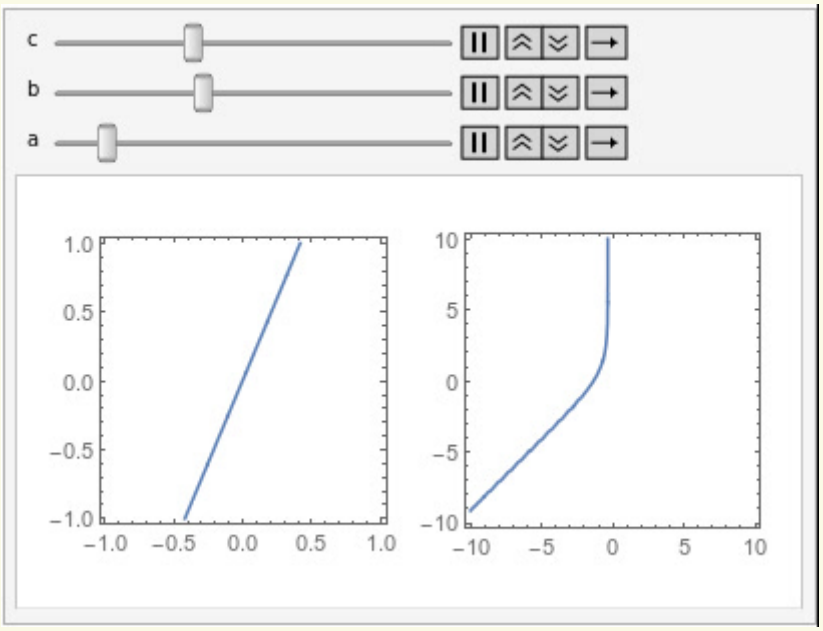
(29)

In other words, linear cuts in the target space, got "squeezed towards the center" in the original space, which turns them into a curve with one bending, this is the effect of the σ^{-1}.

Mathematica code

```
Manipulate[
    GraphicsGrid[{{
        ContourPlot[a x + b y + c == 0, {x, -10, 10}, {y, -10, 10} ],
        ContourPlot[a/(1 + Exp[- k x]) + b/(1 + Exp[- k y]) + c == 0,
            {x, -10, 10}, {y, -10, 10} ]
        }}],
    {{a, .1}, -10, 10, .01},
    {{b, -.2}, -10, 10, .01},
    {{c, 0}, -10, 10, .05},
    {{k, 1}, 0.5, 5}]
```

Readers can try out the shape of these curves in Mathematica. On the left is the straight line in target space, on the right the curve in original space:

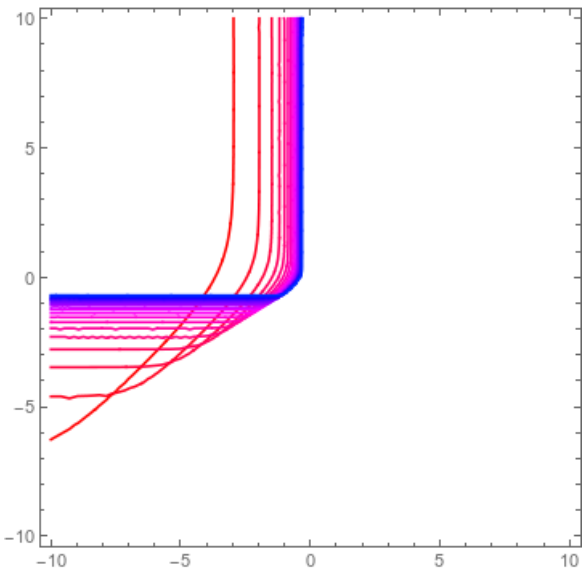


(30)

Notice: if we add a τ parameter into the $\mathcal{S}$ function:

$$\mathcal{S}(x) = \frac{1}{1+e^{-\tau x}} \tag{31}$$

When $\tau \to \infty$, $\mathcal{S}$ turns into $\sqcap$, and these curves turn into "corner" shapes. *:



(32)

This shows that when $\tau \to \infty$, the decision boundary is divided into 4 corner positions.

So much for this, and the most important insight seems to be this: Each neuron represents a binary feature.

---

* Actually, when $\tau \to \infty$, the target space degenerates into 4 vertex points, and those straight lines become **undefined**.
In a previous version of this tutorial, I mistakenly claimed that $\mathcal{S}(x) \to f(x) \equiv x$ as $\tau \to 0$. In fact $\mathcal{S}$ would become a constant function, $f(x) \equiv 1/2$ as $\tau \to 0$.

Multi-layer neural network

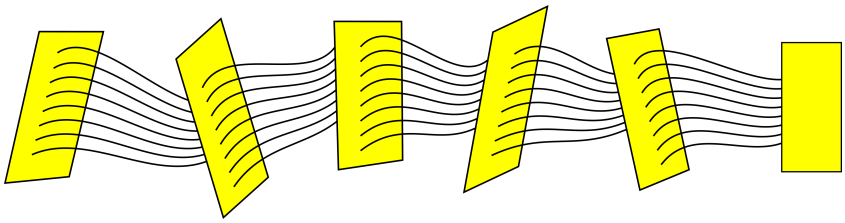
Multi-layer is simply the repetition of single layers:

$$\boxed{\text{output}}\ \boldsymbol{y} = \sigma\overset{1}{W}\ \sigma\overset{2}{W}\ .....\ \sigma\overset{L}{W}\ \boldsymbol{x} \tag{33}$$

L = total number of layers.

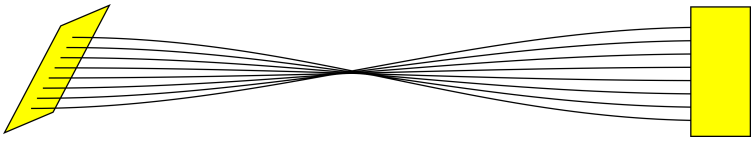
The way to "train" a neural network is by the famous **back-propagation** algorithm, which I won't explain here because it is contained in many standard texts or online tutorials. Basically, the network is given pairs of inputs and their desired outputs. There will be a gap between the network's own output and the desired output, ie, the "error". Back-prop tries to adjust the weights to minimize the total error. It does this via **gradient descent**, a standard optimization algorithm.

In my mind, a multi-layer neural network acts like "pulling and twisting a bunch of noodles": (the diagram is meant to be illustrative only)



(34)

By deforming the σ function, the noodles can be "unwinded" to a simple linear transformation:



(35)

At this point I don't know what is the use of this observation, perhaps it could be related to **braid theory**?

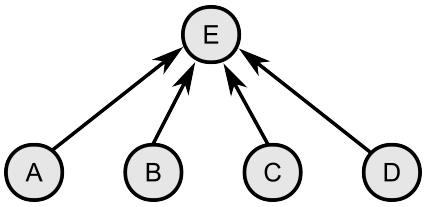
We also observe that the ouput of each neuron does not contain "fuzzy value" information, only *topological* information (ie, which points are neighbors with which points). However, because the neural network is a universal function approximator, we can force it to learn fuzzy values as outputs, if we set the objective function appropriately.

TO-DO: Another thing I want to do is to plot the error surfaces of a multi-layer neural network. From what I've read, these error surfaces seem to have a "staircase-like" appearance. This may explain why during the training of an NN, the error sometimes remains flat, punctuated by *abrupt drops*, which may not be due to the phenomenon of *phase transition*, but is merely a characteristic of the error surface.

Comparison with Bayesian networks

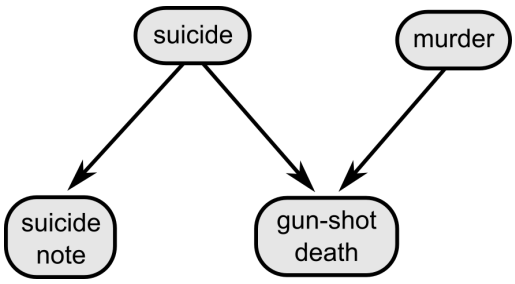
The above is a "geometric interpretation" of neural networks, but there is also a simpler and more direct interpretation: NN as a realization of **propositional logic**. For example, the Bayesian network

below can be regarded as a *neuron* within a neural network; Each node of the Bayesian network represents a proposition:



(36)

But a single neuron's $\int \sum$ operation is too simple, insufficient for calculating probabilities correctly. For example, Bayesian networks are capable of "explaining away", such as in the Bayesian network below, if "suicide note" is true, then the probability of "suicide" increases, and the probability of "murder" decreases:

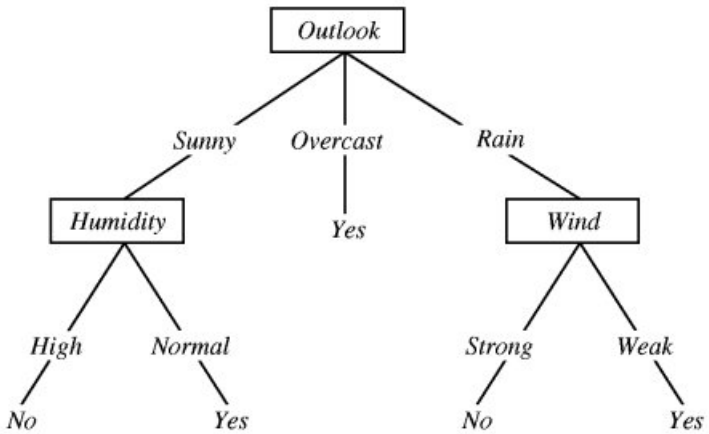


(37)

In other words, the probabilities in a Bayesian network are all connected with each other, they require the use of **belief propagation** algorithms to calculate, which are rather complicated, as they must obey the rules of probability theory. It seems difficult for neural networks to emulate these calculations in a simple manner.

Comparison with decision trees

Let us recall a classic example of decision trees:



(38)

This is the training data:

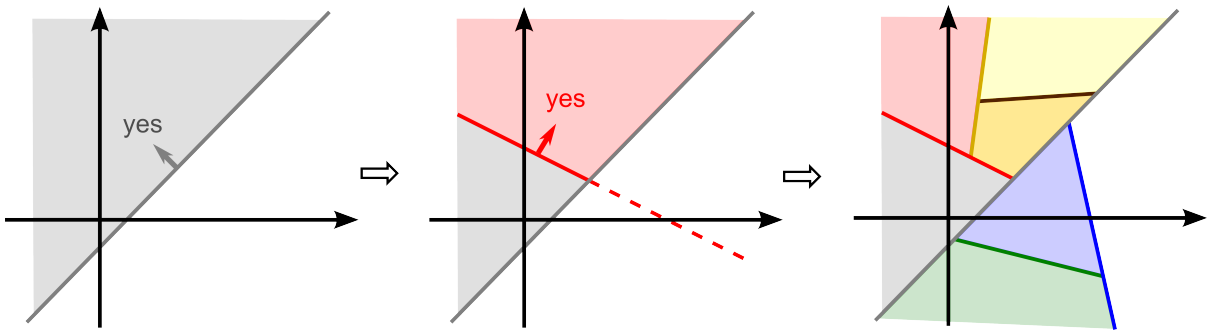
Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

(39)

Decision trees classify things according to **attribute-value** pairs. In order to compare them with neural networks, we assume that each decision tree attribute is a **linear cut**:

$$\sum a_i x_i \overset{?}{>} 0 \tag{40}$$

Decision trees have a special property: after each cut, the 2 halves are processed **separately**, and they have nothing to do with each other subsequently **.



(41)

For example, if the top branch made a cut for "boy / girls", then subsequent cuts for girls may be for "pretty" or "nice figure", whereas for boys may be "tall" and "handsome". The cutting attributes for boys and for girls are different. We could say that decision trees have "shallow cuts", which are different from the "deep cuts" of neural networks.

If we look at the expression for a multi-layer neural network:

$$\boxed{\text{output}}\ \boldsymbol{y} = \sigma\, \overset{1}{W}\ \sigma\, \overset{2}{W}\ \ldots..\ \sigma\, \overset{L}{W}\ \boldsymbol{x} \tag{42}$$

we can see that, for each layer, the results of "classification" are all passed down to the next layer. This is very different from what happens in decision trees: after each "**branch**" of the cut, the branches are processed separately, the essence of the so-called "greedy algorithm". This neural networks are a very special type of classifiers, wherein the higher and lower layers are "intertwined" with many many inseparable connections. This is the reason why neural networks are capable of **representation learning** and **features learning**.

** In diagram (41), cutting the 2D plane many times is not very meaningful, because the output has only the 4 corner vertices. The cuts would be more meaningful when visualized in higher-dimensional space

Relation to propositional logic

Back in the early days of AI, the "father of AI" John McCarthy has already recognized that neural networks are essentially a mimic of simple propositional logic, and he thinks this is a severe limitation. But note that: if the lower layers of the neural network can represent **sub-symbolic** features, then it has the ability to "create" new propositions, and this would be much more powerful and expressive than classical propositional logic.

— The end —