**Machine Learning 26/09/17**

Linear Systems – one of sources goes to neurons

McCullock-Pitts Model of neuron is a linear threshold

**Things being abstracted away from real neurons**

1. Neurons are not instantaneous devices – have complicated systems for whether a synaptic event will cause charge injection

The summing of charges is not quite linear

Also when channel open same amount of ions do not always go in

2. Long delay on output

Spikes travel along the axon at 1-100 metres per second, there are significant delays

3. Neurons get tired

They run low on energy get fatigued, gets harder to cause them to emit spike

4. Ephaptic interactions

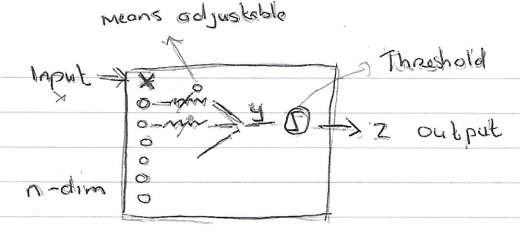
E.g. Brain waves which are voltage changing on scalp, due to activity going on in the brain. When neurons fire spikes you get little voltage change that is detectable there. If those happen at the same time you get something strong enough to detect on the outside. They happen in different frequencies in different places on the scalp. They differ and different times e.g. when sleeping of awake, concentrating or relaxing etc. They are caused by bulk synchronised activity of neurons. It’s thought that it’s not just that they are active together but that the bulk activities cause changes in the bulk potential of different parts of the brain and that these voltage changes in turn can influence the activity of neurons

Some synapses have different neuro transmitters to open different kinds of channels some channels when they open will cause voltage to go up and others cause it to go down. Those that cause it to go up are called excitatory Synapses and those that cause it to go down are called Inhibitory synapses

The weights in the McCullock-Pitt model are arbitrary measures, they can go up or down, can change from negative to positive – not how the hardware is built. The brain as each synapse is either excitatory or inhibitory, can’t change from excitatory to inhibitory all it can do is change how excitatory or inhibitory it is. Long range connections are excitatory e.g. when the signal from the retina hits, it is excitatory whereas short range interactions , have excitatory influences coming from the neurons near them but have inhibitory influences on other neurons

**Rossenplatt – Perceptron**

– take the McCullock-Pitts model and make it work



y = Σw; xi = w . x

z = sign(y)

or if threshold z = sign(y-θ) – normally have threshold as 0 and plug it in at the end

x lives in input space

w are the weights, Sometimes w is a vector which is only part of the adjustable stuff – e.g. w plus the threshold

Weight space (parameter space) includes all the adjustable parameters e.g. all the available w and theta

Another idea is that certain functions can be embodied by the machine. The box embodies some function that give method from input to output – in this case it’s a Boolean function

There will be some mappings from weight space to the function being embodied by box. Setting the knobs becomes one function, change them it becomes another.

Transfer function – transfer from input to output

Has many functions it can embody- each lives in Concept space

Transfer function ∈ Concept space

<w, θ> - all adjustable parameters – live in weight space

W

y = f(xi; W) - transfer function of machine. Takes in input x and gives an output y

y = f(xi;w,θ

f lives in f(., w) € C - C is Concept space

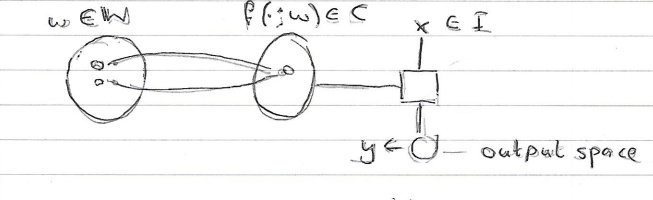
Mapping going on from weight space to concept space

NB Sometimes the mapping is not one to one. In the example the weight space is + 1 dimensional the Concept space in n dimensional, because there is a bit of redundancy in the representation

If we take all the weights and thresholds and multiply them by a scalar, we get the same transfer function.

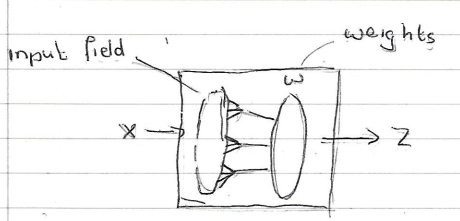
Weight space is generally a little bigger than concept space because of redundancy. We get different weight space using different settings of the parameters

Can by more complicated If the box has a number of different components, if you switch components – another thing that can affect Concept space



Might be more than one point in weight space that maps to the same place in concept space

**3rd Order Perceptron - Rosenblatt**



z = sin(y)

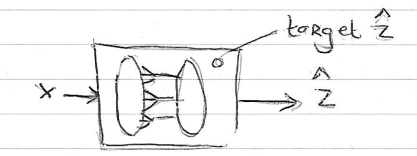
y = w . x = Σw; xi

Slightly different: - have input field and weights and also there’s fixed bits of logic that connect bunch of input fields, combinations of 3 of them – some kind of fixed, nonlinear processing step

e.g. Take combinations of 3 nearby pixels

(Doing pre-processing first is a good idea – have to do it for real world applications)

Had extra input – target could also be + or – 1. You put an input in and get an output out. If it had a learn button, you could press the button. If output is different from the target it would change weights until the output is the same as the target



ẑ -> use hat when you mean approximate when value, z would be the actual value

if ẑ = -1 and -- (Δ = -1)

z = +1

then forAll:

wi <- wi + ξxi - ξ mean small number

if ẑ = z -- (Δ = 0)

then forAll:

do nothing

if ẑ = +1 and --(Δ = +1)

z = -1

then forAll:

wi <- wi - ξxi

**Perceptron Learning Rule**

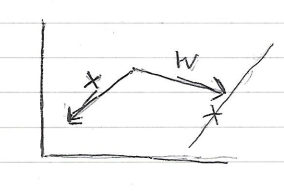
½( ẑ – z) - δ learning rule

w <- wi - ξ ½ (ẑ – z)x

Can write in matrix form W <- w - ξ ½ (ẑ – z)x

- ξ ½ (ẑ – z) is a scalar

Think about this in weight space



**3 Efforts regarding Perceptrons**

1 Build Hardware that embodied it

2. Do Math

3. Apply to real world

Math – first success

**Perceptron Learning Theorem**

If you have a training set, put indexes on it. When you have the choice, call the number of training examples in the set M

M = |Training Set **| -** set of correspondences from input to desired output

Index each – index with superscript, can use ()

{(x(p) \_z(p) }

**Theorem**

If you have a training set and you run the perceptron learning rule on it, it will find a setting of the weights that will get every element of the training set right assuming such a set exists

The times you will have to put training sets through till you get them all right is polynomial time

- has to be able to get them all right

**Perceptron developed to detect tanks**

Training data was provided in the form of lots of images of tanks and images without tanks

When they ran the machine and ran perceptron learning on it, seemed to work

It was then tested on 1000’s of images with and without tanks and passed the tests

It was later discovered that it didn’t work, the reason was that the pictures of the tanks were taken on a sunny day while those without tanks were taken on a dull day. The perceptron was only distinguishing between the levels of light