Brain Modelling

How brains process information.

Nicholas Gale & Stephen Eglen

Brain Modelling.

- Deep Learning is most often applied as a statistical model to understand data.
- Deep Learning is a branch of *theoretical neuroscience* which aims to model brain function and development.
- It is therefore useful to understand some modelling results from theoretical neuroscience.
- We will start by covering some basic brain biology.

The Brain

- A common view is that the brain is a statistical model.
- Takes input as sensory data and output as thoughts/muscle movements/etc.
- A good base to develop models on but the brain is *incredibly* complex and models are *simple*.
- At a basic level it consists of $\approx 10^9$ specialised cells called *neurons*.
- These neurons signal with each other to perform computations.
- Computations are often localised in brain regions called modules e.g. visual cortex.

Neurons

- Neurons thought of in terms of: soma, axon, dendrite.
- Soma is the cell body.
- Axon is a long protrusion connecting to other cells.
- Dendrites connect to axons of other cells and integrate signals.

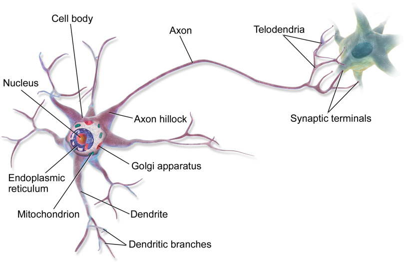


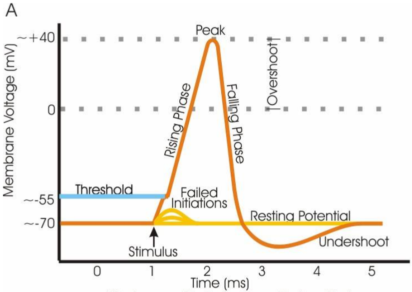
Figure 1: Neuron Anatomy

Membrane Voltage

- Neurons regulate themselves through a membrane voltage.
- This is typically measued in mV and a common baseline is -55mV.
- The voltage is controlled by gated ionic channels: Na, K, etc.
- The membrane voltage can be regulated by current input.

Spiking

- Spiking, or an *action potential*, occurs at a threshold membrane voltage. Referred to as "firing".
- It is a runaway reaction of rapid voltage increase: depolarisation.
- Followed by a voltage decrease: hyper polarisation.
- After an action potential there is a refractory period where the neuron cant fire.



"Schematic" Action Potential

Signals

- An action potential carries this change in voltage down the axon to other cells.
- A neuron can fire many action potentials a second.
- The combined pattern is a signal to other cells e.g. "contract muscle"/"release".

Firing Patterns

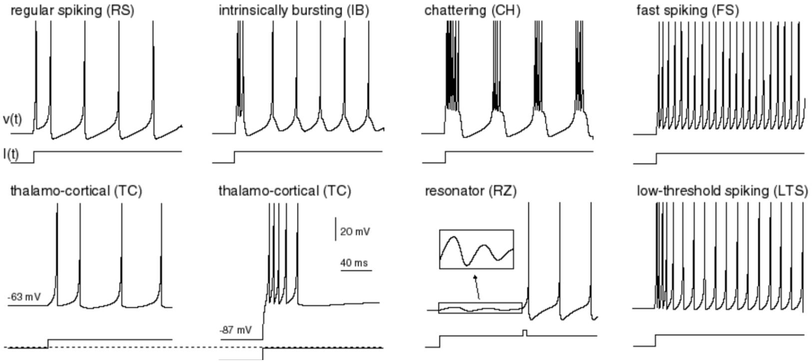


Figure 2: Common Firing Patterns

Synapse

- The axon is connected to a dendrite through a synapse.
- When an action potential reaches the synapse it releases charged neurotransmitters into the synaptic cleft.
- These diffuse through the cleft to bind to the dendrite.
- They modulate the dendrites voltage upward (excitatory) or downward (inhibitory).

Networks

- A neural network is simply the composition of many connected neurons.
- The brain is a large neural network.
- Subdivisions are also networks e.g. auditory cortex, hippocampus etc.
- The inputs to the network and synaptic weights modulate firing patterns of neurons.
- The combined effect is to perform a computation on the inputs.

Networks inspiration

- Reduced models of specific brain networks have been highly successful. *Some* examples:
- Visual system processing: Deep Convolutional Networks.
- Generalised Feed-Forward Networks: Artificial Neural Network (the blueprint).
- Cortical recurrent networks: Hopfield Networks (asscociative memory).
- Visual system development: Optimisation Heuristics (Elastic Net).
- Many more.

Neuron Modelling

- The neurone is a highly complicated structure - even our best models are still reductions.
- The oldest neuron models date to 1901.
- They are almost all based on the law of conductance:

$$C\frac{dV}{dt} = I$$

- Commonly classed as: integrate-and-fire, biophysical, hybrid, or stochastic.

Integrate and Fire

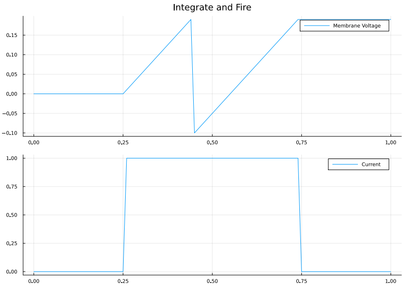
- The oldest form of neuron model.
- Assume neuron conductance is g:

$$\frac{dv}{dt} = gI(t)$$

- If $I(t) = \sum_{t_i \in \text{spike times}} \delta(t - t_i)$ then this "integrates" the spikes into membrane voltage.
- Once a threshold is reached, v is reset to a baseline.

Numerical Implementation:

```julia
using Plots
dt = 0.01; T = collect(0:dt:1);
iT = 0.25 .< T .< 0.75;
 g = 1; thresh = 0.2; reset = -0.
v = similar(T)
for t in 2:length(T)
    dv = g * iT[t]
    v[t] = v[t-1] + dt * dv
    if v[t] > thresh
        v[t] = reset
    end
end
```



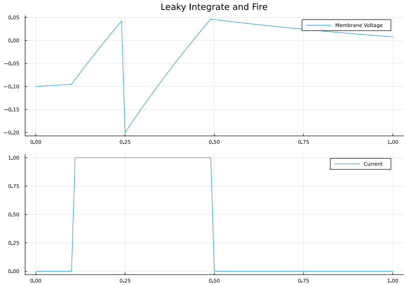
Leaky Integrate and Fire

- A slightly more sophisticated model recognises that a neuron tends to "rest" at a baseline.
- If it doesn't fire its membrane voltage decays through ion channels diffusion.
- To model this we assume a resting voltage of v_r and a differential equation:

$$\frac{dv}{dt} = -g(v - v_r) + gI(t)$$

Numerical Implementation:

```
dt = 0.01; T = collect(0:dt:1);
iT = 0.1 .< T .< 0.5;
g = 1; thresh = 0.05; reset = -0.
vr = -0.05;
v = zeros(length(T)) .- 0.1
for t in 2:length(T)
    dv = g * iT[t] - (v[t-1]-vr)
    v[t] = v[t-1] + dt * dv
    if v[t] > thresh
        v[t] = reset
    end
end
```



Hodgkin-Huxley

- The Hodgkin-Huxley is a jewel of theoretical neuroscience (and modelling more generally).
- It explicitly models Na and K ionic channels with condutances g_{Na} and g_k and a leaky channel g_L.
- It models the response of these channels to a membrane voltage through response variables: n, m, h.
- It does not require a "hard-reset" and is able to accurately predict neuron behaviour quantitatively.

Hodgkin-Huxley Definition

- The differential equations are as follows:

$$C\frac{dv}{dt} = g_K n_1^4 (v_K - v) + g_{Na} n_2^3 n_3 (v_{Na} - v) + g_L (v_L - v) + I(t)$$

$$\frac{dn}{dt} = \alpha_{n_i}(v)(1 - n_i) + \beta_{n_i}(v) n_i$$

- The functions α_j, β_j take the generic form:

$$\frac{A_j(v - B_j)}{\exp\left(\frac{v - B_p}{C_p}\right) - D_p}$$

- The parameters can be found by fitting to neural voltage data.

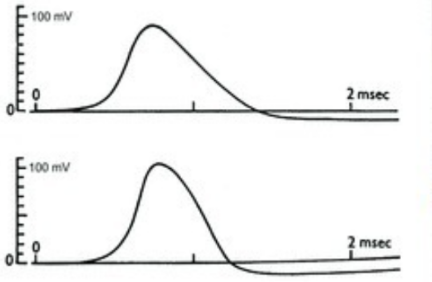


Figure 3: Membrane voltage of giant squid axon (top) against the model prediction (bottom)

Hybrid Models

- The Hodgkin-Huxley model is *accurate* and has been extended for even more accuracy, but is *expensive*.
- The integrate-and-fire models are *cheap*.
- Hybrid models reduce the biophysical HH type models to blend efficiency and accuracy.
- They usually consist of a voltage (v) and auxillary (u) variable. They can be useful in analtyical and computational frameworks.
- The best in class is the Izhekivich model.

Hybrid Model Fit

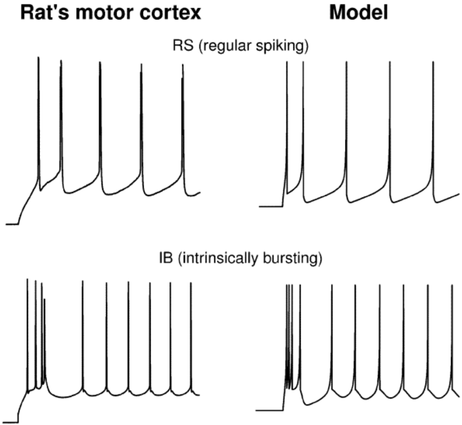


Figure 4: Izhekivich model spiking predictions compared to rat cortical neurons

Firing Rates

- All of the models presented can generate various classes of behaviour: quiet, single-spikes, periodic spikes.
- Under constant current conditions periodic spiking can be described by the firing rate (Hz)
- This is an important biological measurement and computational reduction.
- A school of thought believes that the rate encodes the neurons computation: rate based computation.
- There is evidence for this but it is simplistic. It does however form the basis of Deep Learning models.

Stochastic

- The final class of model is a stochastic model.
- It asserts that neuron spiking is distributed as a Poisson process parameterised by firing rate r:

$$X \sim \text{Poisson}(r)$$

Implementation

- A spike train is created by choosing a rate r and a time interval dt.
- At the Nth time step (time $t_0 + Ndt$) sample a uniform random number p.
- If $p < rdt$ then record a spike.
- This is the cheapest spiking model to implement.

Inhomogeneous Poisson

- The inhomogenous process process allows $r = r(t)$ if the rate changes slowly.
$$X \sim \mathrm{Poisson}(r(t)$$
- This process captures most statistical properties of real neurons.
- Stochastic process are a cheap way to generate spike data.

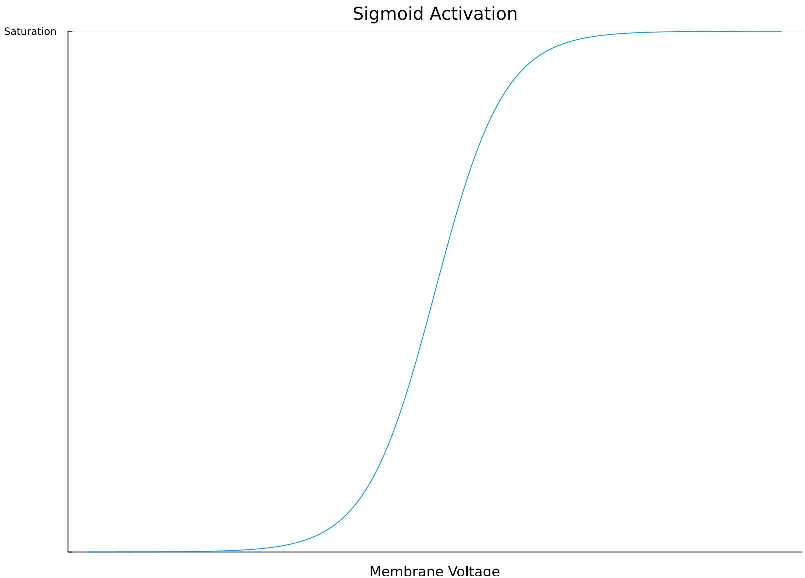
Activation Functions

- A probabilistic intepretation leads us to relate rates/probabilities with membrane voltage
- An activation function measures the "activity" of a neuron as a response to membrane voltage.
- The most biologically accurate function is the sigmoid.

Sigmoid Activation

- Let the maximum firing rate of a neurone be Q; its threshold be θ; and slope-response to voltage be β
- The sigmoid function s is given by:

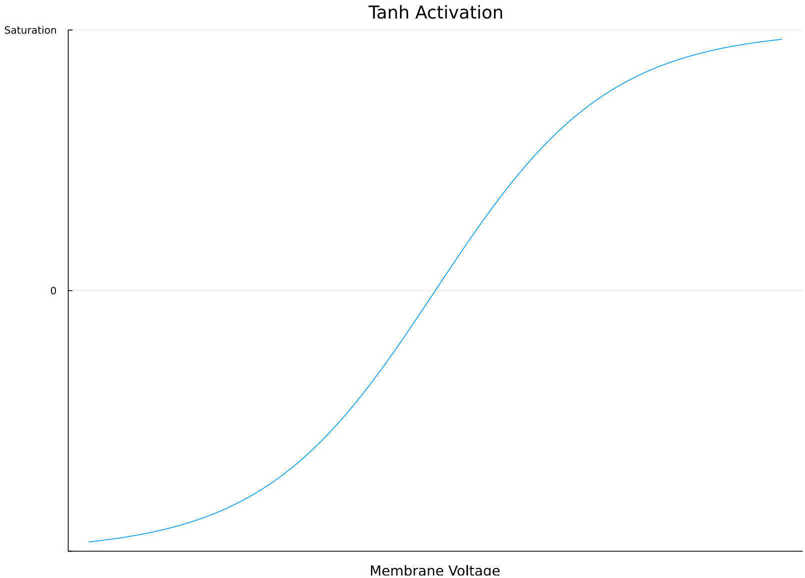
$$s(x; \beta, Q, \theta) = \frac{Q}{1 + \exp\left(-\frac{x-\theta}{\beta}\right)}$$


Sigmoid Activation
Saturation
Membrane Voltage

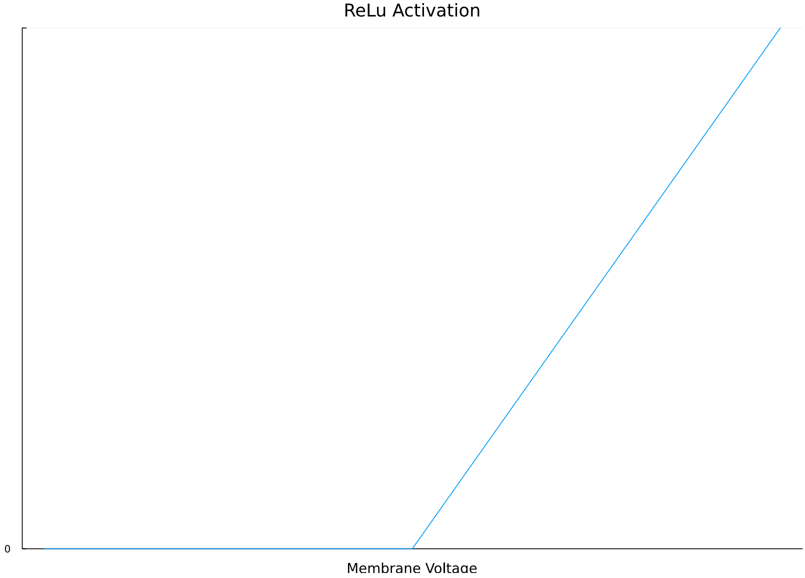
Common Activation Functions

- Other activation functions are commonly chosen.
- These are less accurate but have nice mathematical/computational properties.
- Tanh(x): a scaled sigmoid to be odd around the origin.
- ramp(x): a computationally cheap approximation. Also called Rectified Linear Unit (relu)
- Heaviside(x): a simple on/off intepretation that is useful in proofs.

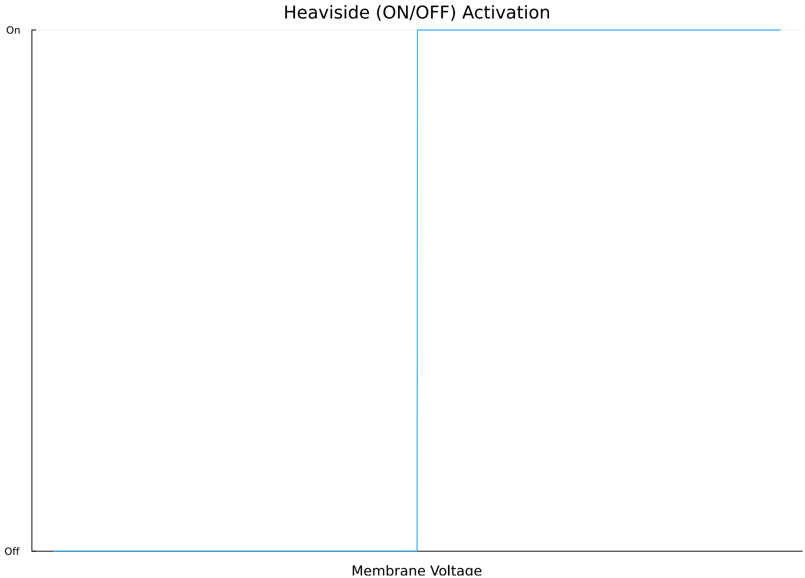
Tanh



Ramp/Rectified Linear Unit (ReLu)



Heaviside



Neural Network Models

- We have built up a sophisticated repotoire of neuron activity models.
- We would like to build them into something more useful.
- To do this we incoporate them into a dynamical network model.

Networks

- A network is simply a graph with nodes $i \in 1 : N$ and edges W_{ij}.
- A dynamical network model is composed of internal dynamics of the node and contributions from the edges.
- It typically takes the form:

$$\frac{dv_i}{dt} = f(v_i, t) + \sum_{j=1} NW_{ij} g(v_j, t)$$

Dendrites

- A neural network model replaces f with the activation dynamics and W_{ij} with the contributions from other neurons.
- W_{ij} have a physical intepretation: they are dendritic weights.
- g represents some response function such as spiking.
- This is the most general form of a neural network.
- *All* of the above neurone models will be able to learn an input response.
- The most common form is a linear response to weights passed through an some activation function.

The Hopfield Network

- The Hopfield Network is a seminal work in neural network theory.
- It is a good statistical classifier.
- It has a naturally intepretable structure.
- It has a biologically derived learning rule and provides explanations for biological phenemona.
- It therefore ticks *all* the boxes.

The Hopfield Network Setup:

- In the Hopfield model a pool of N cortical neurons are all connected to each other.
- The neurons are indexed by i and the weights by W_{ij}. The inputs are given to neurons as $u_i^0 = I_i$.
- The neurons state is evolved as

$$u_i^{t+1} = \mathsf{sign}(\sum_j Wiju_j^t)$$

- The sign function classifies neurons as active/inactive and the state is evolved until it is steady.
- The steady state can be compared to a hashing function to classify an input.

The Hebb rule: Training

- The Hopfield network uses the Hebb rule to train its weights.
- The Hebb rules states: neurons that fire together wire together. This means coactive neurons strengthen; otherwise they decay.
- To encode an input pattern v we take the autocovariance as the weight change i.e. the Hebb rule:

$$\Delta W_{ij} = v_i v_j$$

- The autocovariance relationship ensures that the steady state of the pattern v is v itself.
- To train M such patterns we simply take the mean of the linear combinations of the Hebb rule.
- We also hash the patterns for classification later.

Hopfield Network Training

```julia
using Random
delW(v) = sign.(v) .* sign.(v)'
function constructW(data)
    W = zeros(length(data[1]), length(data[1]))
    for v in data
        W .+= delW(v)
    end
    return W ./ length(data)
end
data = [sign.(rand([-0.2, 0.8], 784)) for i in 1:10]
labels = [randstring(10) for i in 1:10]
hash = Dict(zip(data, labels))
trained = constructW(data)
```

Partial Input

- The Hopfield network can then be used to query inputs and classify them.
- We find we can delete substantial portions of vectors and still classify them correctly.
- We can also classify vectors that are “similar”.

Partial Input: Prediction

```
class_predict (generic function with 1 method)
```

```
corrupt(c) = map(x -> (rand()>0.5) ? x : -sign(x) * rand(), c)
for i in data
    pred = class_predict(trained, hash, corrupt(i))
    cl = hash[i]
    println((pred, cl, pred == cl))
end
```

```
("XXzNkTSiux", "XXzNkTSiux", true)
("NIFF5t2MzF", "NIFF5t2MzF", true)
("v0sEe2pVeb", "v0sEe2pVeb", true)
("bHvsrCx4zU", "bHvsrCx4zU", true)
("DtP98i3Q1Y", "DtP98i3Q1Y", true)
("Q95mIpNqZ2", "Q95mIpNqZ2", true)
("MSmW4Ic13J", "MSmW4Ic13J", true)
("PhxE4qllhq", "PhxE4qllhq", true)
("iupUytKyzj", "iupUytKyzj", true)
("2Ua7WzICnz", "2Ua7WzICnz", true)
```

Robustness

- The network is remarkably robust.
- We can delete large swathes of parameters and it retains classification power.

```
h(x) = x .* (x > 0)
deletion_fraction = 0.5
stroke_trained = trained .* h.(rand(size(trained)...) .- deletio
for i in data
    pred = class_predict(stroke_trained, hash, corrupt(i))
    cl = hash[i]
    println((pred, cl, pred == cl))
end
```

```
("XXzNkTSiux", "XXzNkTSiux", true)
("NIFF5t2MzF", "NIFF5t2MzF", true)
("v0sEe2pVeb", "v0sEe2pVeb", true)
("bHvsrCx4zU", "bHvsrCx4zU", true)
("DtP98i3Q1Y", "DtP98i3Q1Y", true)
("O95mIpNgZ2", "O95mIpNgZ2", true)
```

Memory Capacity

- Why not just build a network to store “all” patterns.
- After a certain number of new patterns it begins to erase old patterns.
- The network has a capacity limit.
- This tendency to erase learned patterns is known as *catastrophic forgetting*.

How it works.

- The model constructs an energy function (Lypaunov) which the trained patterns are local minima.
- They correspond to spin-glass states in an Ising model.
- These local minima form attractive basins in the energy landscape.
- Patterns that are “close” to trained patterns fall into these basins and return the trained states as output.
- Network can be tricked by spurious patterns: combinations of trained states.

What does it tell us?

- In addition to being an excellent classifier the Hopfield network allows us to make biological insights.
- It validates the Hebb rule as a memory learning rule.
- It offers an explanation for asscociative memory: things can “ring a bell”.
- It allows us to understand the effects of partial recall and stroke.
- It implies that there are limits to cortical memory.

The Perceptron

- The neural network dynamics on a single neurone with dendritic inputs W, an activation function f and a resting potential b look like:

$$y = f(Wx + b)$$

- This is called a perceptron and can be used to classify things in binary format using a threshold.
- Multiple perceptrons can be encoded in a vector with a weight matrix and vector biases.

Perceptron Training

- The perceptron model can be trained by a routine that moves inputs closer to their target.
- This is also routed in the brain: attention signals can mediate desired outputs.
- The routine for training a regressor x_t with ouput xi_t and target output y_t is:

$$W_{ij}(t+1) = W_{ij}(t) + r(x_i - y_i)v_j$$

- The learning rate is dictated by r.
- This is nothing more than minimising least-square-error on linear regressors
- This is true for any activation function that is monotonic.

Perception Training.

```
function train_epoch(class1, class2, W, b, r, thresh)
    # class 1 < regression, class 2 > regression
    for i = 1:(length(class1[1]) + length(class2[1]))
        if i <= length(class1[1])
            datum = [class1[1][i], class1[2][i]]
        else
            datum = [class2[1][i], class2[2][i]]
        end
        pred = W * datum .+ b
        err = (pred[1] > thresh)
        W = W .- r .* err .* datum'
        b = b .- r .* err
    end

    return W, b
end
```

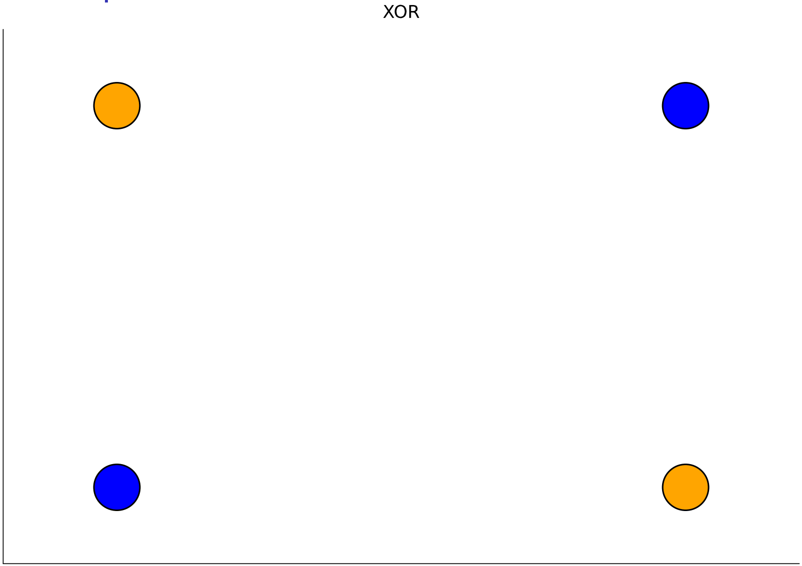
```
train_epoch (generic function with 1 method)
```

The XOR problem.

- The perceptron model was very popular but it is not *general*.
- It can only do linear regression, which is interesting, but not brain-like.
- This was highlighted by the XOR problem: a perceptron model of the XOR gate.
- XOR is given by:

$$(00) \mapsto 0, (01) \mapsto 1, (10) \mapsto 1, (11) \mapsto 0$$

The XOR problem visualised.



- The false values are indicated in blue and the true values in orange.

Two layers: solving XOR

- To solve the XOR problem we need to augment the perceptron.
- We do this by having two layers of perceptrons and feed the output of one into the other.
- We *need* the activation function to be non-linear (otherwise it will reduce to a single perceptron layer).
- This is no longer performs linear regression.

The solution:

```julia
relu(x) = x .* (x .> 0)
layer1(x) = relu([1 -1; -1 1] * x)
layer2(x) = [1 1] * x
xor(x) = layer2(layer1(x))
xor.([[0,0], [0,1], [1,0], [1,1]])
```

```
4-element Vector{Vector{Int64}}:
 [0]
 [1]
 [1]
 [0]
```

Multi-Layer Perceptrons: General Artificial Neural Networks.

- When multiple layers of perceptrons are chained together they are called: *multi-layer perceptrons* (MLPs).
- These are the most general form of *artificial neural networks* (ANNs).

- They can be used to model *any* function and are thus universal models.
- When they have many layers they are considered "deep" thus the name Deep Learning.
- We need a way to teach them.

Summary.

- Biological models of neurons exist from highly realistic and expensive, to cheap but less accurate.
- Neurons can be modelled as a network and this can perform any function (with training).
- The Hopfield network is a simple setup with a biological training rule that explains much of brain function.
- The perceptron is an easy to train biologically inspired linear regression model.
- The multi-layer perceptron is an all-purpose statistical model.