Deep Learning – Neural Nets

The McCulloch-Pitts Perceptron, 1943

- "Perception" and "Electronic"
- A very simplified model of how a real neuron works
- "Perceptron" and "Neuron" used almost interchangably,
  but Perceptron is a little bit old fashioned now
- Any number of inputs, a_i
  Sometimes on/off, sometimes continuous
- Each input has an associated weight, w_i
  Changing the weights is how it learns
- Output or State computed from weighted inputs
  $\sum a_i \times w_i$
- But a non-linear Activation Function is applied to that sum
  $f(\sum a_i \times w_i)$
- Could be (but usually isn't) a cut-off function
  $f(x) =$ if $x < k$ then 0 else 1
  each perceptron can have its own value for k
- Or the Sigmoid (or Logistic) function, popular
  $f(x) = 1 / (1 + e^{-x})$
  easy to differentiate, that will turn out to be useful
- Or the Rectified Linear Unit (ReLU) function
  $f(x) = \max(0, x)$
- Or the Softplus function, a continuous approximation to ReLU
  $f(x) = \log_e(1 + e^x)$
- Or the Hyperbolic Tangent function tanh
  $f(x) = (e^{2x} - 1) / (e^{2x} + 1)$
  but that gives a range of -1 to +1
- There can also be a Bias
  Typically a constant value -1 or +1
  that is treated as an extra input with its own weight
  it can shift the curve given by f to the left or right

Layers

- One perceptron recognizes a straight-line/plane/hyperplane
  Separating one set of possible inputs from the other
- Originally arranged in a single layer
  Usually all receiving the same inputs from sensors
  And all recognizing different Features
- But that can't recognize such a simple thing as exclusive or
- Now usually arranged in multiple layers
  This is what Deep Learning refers to
  There is an input layer and an output layer,
  any others are called Hidden Layers
- Just two layers can compute any continuous function
  As close as you like, just add more perceptrons

- Maybe softmax for final outputs when classifying
  $e^{thisoutput} / \sum e^{alloutputs}$
  Suppose the actual outputs are 0.3, -0.4, 1.2, 0.2
  the softmaxes are 0.21, 0.10, 0.51, 0.19
  can use as probabilities, biggest tends to get exaggerated
- Big things, vision, language, etc. can use many layers
- Neurons in real brains are not really arranged in layers
  And are definitely not Fully Connected either

Training with only two layers, inputs and outputs

- Improve performance by adjusting weights throughout the network
- Diminishing Returns – stop when it isn't making much difference any more
- Use a training set of course - large set of pairs (inputs, desired outputs)
- adjust each link according to the amount of error it was responsible for
- For each training point, work out the Local Error of the entire network
  Just the difference between the actual output and the correct output
  squared of course, negative errors are just as bad as positive ones
  If there are multiple outputs, just average all their errors
  That is Mean Squared Error, MSE
  Sometimes Root Mean Square, RMS, is used instead
- Repeat that for the entire training set, averaging all the errors
- If that average error is small enough
  the network has Converged. Training is over.