

Neural Nets and the Perceptron

Daniel Lawson — University of Bristol

Lecture 07.1 (v2.0.0)

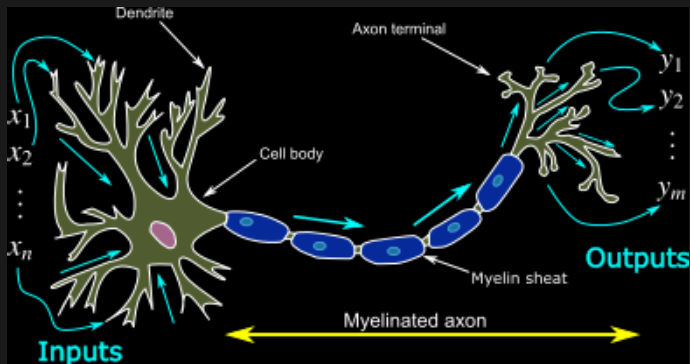
Signposting

- ▶ This Block is split into two Lectures:
 - ▶ 07.1 (this lecture) on the theory
 - ▶ 07.2 on practicalities
- ▶ This is Part 1, which covers:
 - ▶ Introduction
 - ▶ Neurons
 - ▶ Single layer perceptron
 - ▶ Learning algorithms

Questions

- ▶ What makes a neural network deep?
- ▶ Does deep matter?
- ▶ How can we learn parameters for a neural net?
- ▶ What are the most important types of neural net?

Neurons



- ▶ Dendrites take inputs
- ▶ Axons fire on activation
- ▶ Form a **dynamical system**

Artificial Neurons

- ▶ Take a number of input signals
- ▶ Activation function transforms to output
- ▶ Output sent as input to downstream neurons
- ▶ (Typically) constructed to form a **directed system** for learning

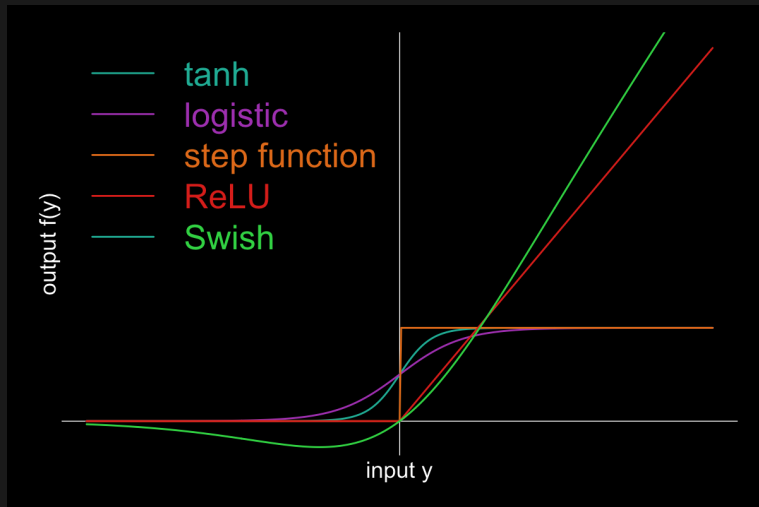
Activation functions

- ▶ Neuron i is modelled as:
 - ▶ A nonlinear **activation function** f :
 - ▶ a base rate $W_{0,i}$,
 - ▶ and weights $W_{j,i}$ for each input neuron a_j with output x_{a_j} :

$$f \left(W_{0,i} + \sum_{j=1} W_{j,i} x_{a_j} \right),$$

- ▶ f is a mapping $\mathbb{R} \rightarrow [r_{min}, r_{max}]$ (which may not be bounded).
- ▶ There are many common choices, e.g.:
 - ▶ tanh: $f(y) = (1 + \tanh(y)) / 2$
 - ▶ logistic: $f(y) = 1 / (1 + e^{-y})$
 - ▶ Step function: $f(y) = \mathbb{I}(y > 0)$
 - ▶ Rectified linear unit (ReLU): $f(y) = \mathbb{I}(y > 0)y$

Activation functions



Activation functions

- ▶ The important features of activation functions are:
 - ▶ **Non-linearity**. A deep neural network can be trivially replicated by a one layer neural network if the activations are linear.
 - ▶ **Derivatives**. Learning requires evaluating derivatives, which should be *cheap*, and *informative*.
 - ▶ **Smoothness**. Simple discontinuities can be handled, complex ones make learning slow.

Activation functions in practice

- ▶ ReLU contains the important complexity whilst being very fast to learn;
- ▶ It may exhibit convergence problems when $y \ll 0$;
- ▶ For small networks, complex activation helps.
- ▶ A notable modern alternative is **Swish**¹:
 - ▶ $f(y) = y / (1 + \exp(-\beta y))$
 - ▶ **ReLU-like**: Converges to zero for $x \rightarrow -\infty$ and to x for $x \rightarrow \infty$
 - ▶ Has **unbounded derivative** for $x < 0$ so learning still works
 - ▶ Strangely, monotonicity seems not to be important?

¹Ramachandran, Zoph and Le **Searching for Activation Functions**

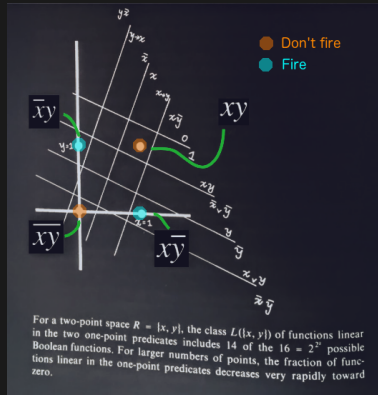
Logical functions

- ▶ Every boolean function can be implemented by a neural network².
- ▶ For simplicity $f(x \leq 0) = 0$, and $f(x > 0) = 1$, i.e. the neuron “fires” on activation. Then, the following can be implemented on a single node:
 - ▶ AND: $f(x_1, x_2) = -1.5 + x_1 + x_2$
 - ▶ OR: $f(x_1, x_2) = -0.5 + x_1 + x_2$
 - ▶ NOT: $f(x_1) = 0.5 - x_1$
- ▶ Neural networks with more general activation functions can still implement these functions.

²McCulloch and Pitts (1943) A logical calculus of the ideas immanent in nervous activity

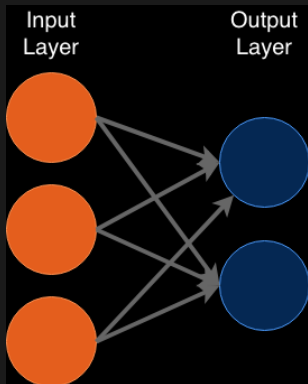
Logical function problems

- ▶ But not every function can be implemented in a single layer perceptron³:
 - ▶ XOR: only x_1 or x_2 can be active



³Minsky and Papert 1969 Perceptrons

Single Layer perceptron (SLP)



- ▶ Has just two layers:
 - ▶ data layer (e.g. features)
 - ▶ output layer (e.g. classes)
- ▶ No **hidden** layers!
- ▶ Weights learned
- ▶ Making a linear classification rule

Mathematical description of SLP

- ▶ N Inputs x_i and M outputs y_j
- ▶ Activation function f and with weights W_{ij} :

$$f(\mathbf{x}) = f\left(W_{0j} + \sum_{i=1}^N W_{ij}x_i\right)$$

- ▶ W_{0j} allows for an offset (mean) in the activation, just like in linear regression
- ▶ Loss is the square error over all output variables j :

$$\begin{aligned} L(W) &= \sum_{j=1}^M L_j = \sum_{j=1}^M \left[y_j - f\left(W_{0j} + \sum_{i=1}^N W_{ij}x_i\right) \right]^2 \\ &= \sum_{j=1}^M \delta_{ij}^2(\mathbf{w}_j) \end{aligned}$$

- ▶ $\delta_{ij}(\mathbf{w}_j)$ is the error for input i output j .

Learning through Gradient Descent

- ▶ Learn through Gradient Descent:
 - ▶ i.e. Differentiate the loss with respect to the **weights** for $i = 0, \dots, N$:

$$\nabla_W L = \left(\frac{\partial L}{\partial W_{10}}, \dots, \frac{\partial L}{\partial W_{ij}}, \dots, \frac{\partial L}{\partial W_{NM}} \right)^T$$

- ▶ where:

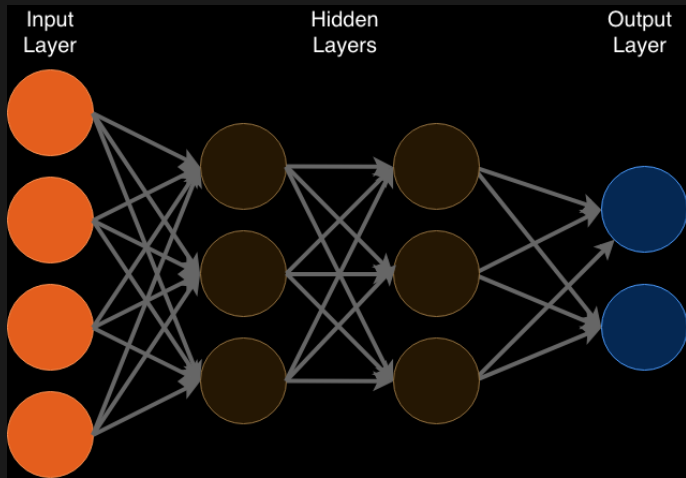
$$\frac{\partial L}{\partial W_{ij}} = \frac{\partial L}{\partial f} \frac{\partial f}{\partial W_{ij}} = -2\delta_{ij} \frac{\partial f}{\partial W_{ij}},$$

- ▶ Leading to the update rule:

$$W_{ij} \leftarrow W_{ij} + \alpha \frac{\partial f}{\partial W_{ij}} \delta_{ij}$$

- ▶ We are taking a step of size α in a direction towards the multivariate **minima of the loss**
- ▶ Choose step size α to take steps that move *fast enough* whilst not *overshooting*.
- ▶ In practice α is learned adaptively.

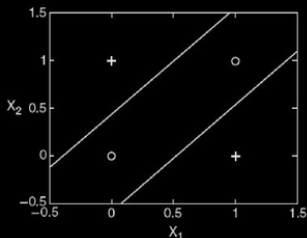
Multilayer Perceptrons / Feed Forward Neural Networks



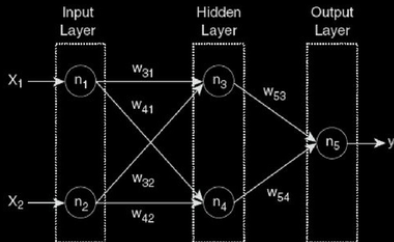
Multilayer Perceptrons / Feed Forward Neural Networks

- ▶ A Neural Network's power is in **hidden layers**
 - ▶ Hidden layers can be treated exactly as the layers we have observed
 - ▶ Maths allowing modularly that is transformative
- ▶ **Architecture** choices include the number of layers and the connectedness:
 - ▶ Completely connected layers?
 - ▶ Locality towards data?
 - ▶ Number of neurons in each layer?
- ▶ These choices are somewhat manual and define your **model**
- ▶ Architecture is robust, i.e. many choices will lead to similar predictions. . .
- ▶ But they are **not** arbitrary!

Universal Approximation Theorem



(a) Decision boundary.



(b) Neural network topology.

Figure 5.19. A two-layer, feed-forward neural network for the XOR problem.

- ▶ Any⁴ function of n inputs can be approximated
- ▶ By using **non-linear** activation functions (e.g. ReLU)
- ▶ Using a **single hidden layer**, with an **exponential width** (number of nodes, scale with n)
- ▶ Or a (linear in n) **deep network with finite width**

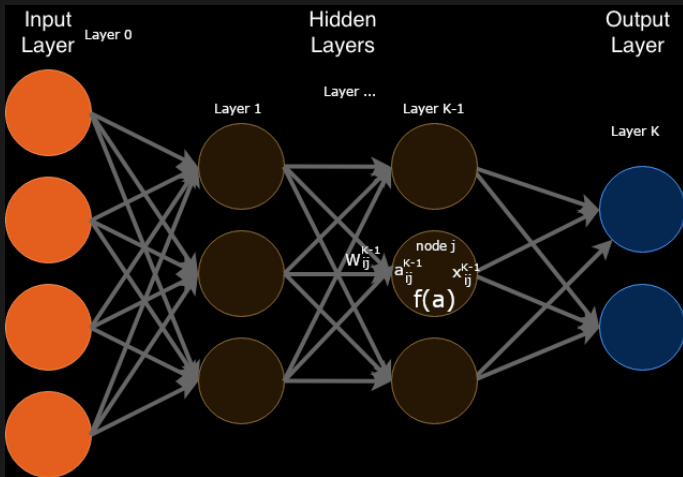
⁴continuous, compact function on \mathbb{R}^n

Back Propagation

- ▶ Learning Neural networks was an art until **back propagation** was discovered⁵.
- ▶ This is a method to compute all derivatives of all weights, exactly and efficiently.
- ▶ Notation:
 - ▶ Index the current layer as k (of K) with node labels i , the next layer with labels j .
 - ▶ Activation function $x_j^k = f(a_j^k)$
 - ▶ $a_j^k = W_{0j}^k + \sum_{i=1}^{n_k} W_{ij}^k x_i^k$
- ▶ Output layer: W_{ij}^K is learned as a Single Layer Perceptron
- ▶ Work backwards from there...

⁵Hecht-Nielsen, Robert. "Theory of the backpropagation neural network." Neural networks for perception. Academic Press, 1992. 65-93.

Backpropagation network



Back Propagation

- ▶ Hidden layers: back-propagate the error from the **next layer** to the **current**, using the chain rule:

$$\frac{\partial L}{\partial W_{ij}^k} = \sum_{j=1}^{n_{(k+1)}} \frac{\partial L}{\partial x_j^{(k+1)}} \frac{\partial x_j^{(k+1)}}{\partial a_{ij}^{(k+1)}} \frac{\partial a_j^{(k+1)}}{\partial W_{ij}^k}$$

- ▶ i.e. we compute the activation function for one layer as a (sum over) two components:

- ▶ **error** : $\delta_j^{k+1} = \frac{\partial L}{\partial x_j^{(k+1)}}$

- ▶ **response** : $\frac{\partial x_j^{(k+1)}}{\partial a_{ij}^{(k+1)}} = \frac{\partial f(a)}{\partial a}$

- ▶ **response rate** : $\frac{\partial a_j^{(k+1)}}{\partial W_{ij}^k}$

- ▶ The last two are often combined, but this representation separates the activation function from the weights.

Stochastic Gradient Descent

- ▶ **Gradient Descent** is just the beginning. It is appropriate for:
 1. **Smooth** or **convex** error functions, so that we do not become trapped in a local optima;
 2. **Small data regimes**, where we can afford to compute the entire gradient every update.
- ▶ **Stochastic Gradient Descent** addresses local minima and computational cost together.
 - ▶ It uses **mini-batches** of data for a gradient update.
 - ▶ This makes each update **random**, creating a type of **annealing** in the algorithm:
 - ▶ We can take large random steps when we are far from the optima (large step size),
 - ▶ And much shorter and hence on average reliable steps when we are closer (small step size).

Interpreting classifier output

- ▶ Neural networks output a set of **activations**
- ▶ It is standard to apply **softmax** $p(\mathbf{z}) : \mathcal{R}^n \rightarrow [0, 1]$ s.t.
 $\sum_{i=1}^n z_i = 1$:

$$p(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

- ▶ This interprets the activation as a log-likelihood
- ▶ This is **almost always wrong**

Interpreting classifier output

- ▶ Various sophisticated approaches are available:
 - ▶ e.g. Mixture Density Networks⁶
 - ▶ Calibrate probabilities in a “post processing” layer⁷
- ▶ Neural Networks are **not** (normally) approximating probabilities. They are predicting data, or equivalently, predicting decisions.
 - ▶ e.g. A NN driving a car doesn't care about the probability of a person being in the screen.
 - ▶ It cares about the Loss function, which in this case would be expressed in terms of **actions**.

⁶Bishop 1994 **Mixture Density Networks**

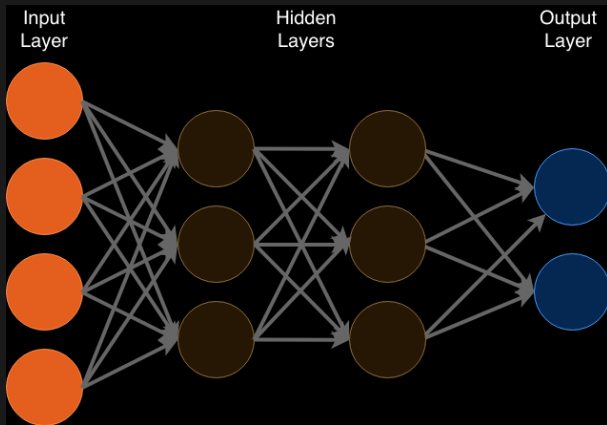
⁷Kull et al 2019 NeurIPS **Beyond temperature scaling: Obtaining well-calibrated multiclass probabilities with Dirichlet calibration**

Some types of neural network

- ▶ Feed-forward
- ▶ Convolutional
- ▶ Recurrent
- ▶ Recursive
- ▶ Auto-encoders
- ▶ ...

Feed forward neural network

- This is the Neural Network that you know. It is acyclic.



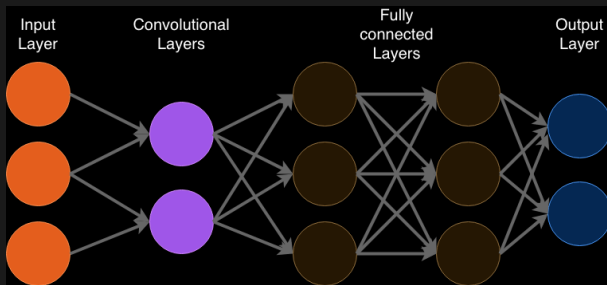
Feed forward neural network

- ▶ The feed forward neural network is a **universal approximator**
- ▶ It can therefore be used as a **component** of a NN to compute **any** function $\mathbf{y} = f(\mathbf{x})$
- ▶ This can include:
 - ▶ **Likelihoods**, so making **probabilistic** predictions
 - ▶ **Derivatives**, (which are evaluated in the feed-forward step!)
 - ▶ And anything else we can imagine.
- ▶ Learning f can be complex, though many papers provide their network.
- ▶ Although all functions are approximable, not all behave nicely.
 - ▶ For example, densities seem hard to approximate whilst cumulative distribution functions behave better⁸.

⁸Chilinski and Silva **Neural Likelihoods via Cumulative Distribution Functions**

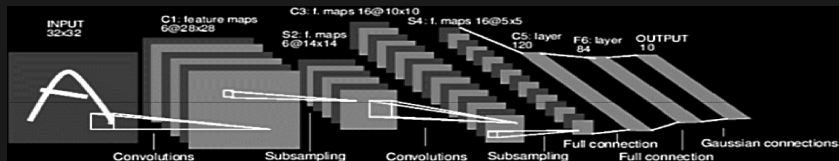
Convolutional neural network

- ▶ This is a feed-forward network that has carefully designed layers for constructing **known features**, such as local averaging.



- ▶ Choosing CNN architecture is **choosing a model**
- ▶ It should reflect known structure, e.g. locality, exchangeability, etc

Convolutional neural network

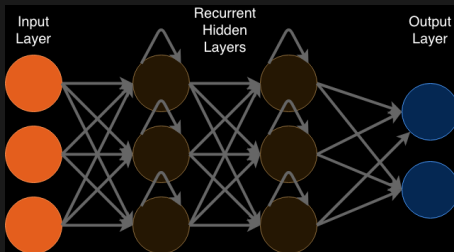


- ▶ CNNs are a core part of image processing⁹
- ▶ They scan an image, constructing **features**
- ▶ Different convolutions can create different features, including:
 - ▶ Larger objects
 - ▶ Edges
 - ▶ Presence/absence of either via max-pooling

⁹Albawi, Mohammed and Al-Zawi **Understanding of a convolutional neural network**

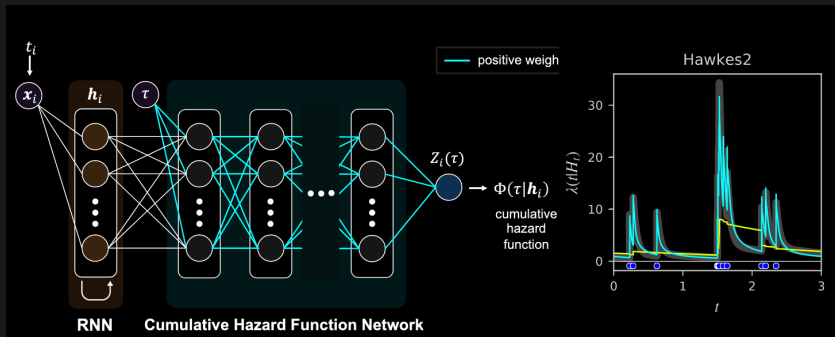
Recurrent Neural Network

- ▶ This is a network containing cycles, which allows for “memory” and potentially chaotic behavior.



- ▶ Training is hard; uses a special algorithm: “*causal recursive backpropagation*” which mitigates the disconnect between error and weights in standard algorithms. . .

Recurrent Neural Network for Point Processes



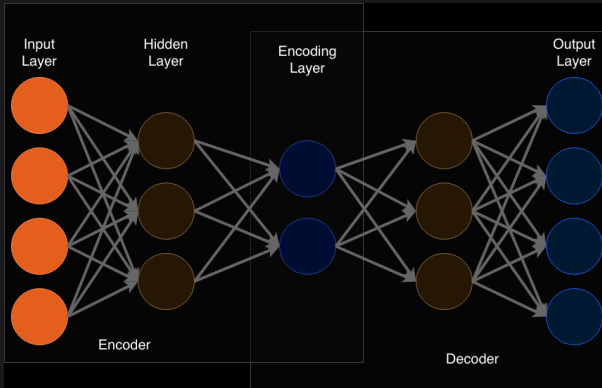
- ▶ An RNN acts as a “memory” for an arbitrary history¹⁰
- ▶ A CNN acts as a universal approximator to the CDF
- ▶ This is translated into the Likelihood of the data by back-propagation differentiation

¹⁰Omi, Ueda and Aihara Fully Neural Network based Model for General Temporal Point Processes

Recurrent Neural Network

- ▶ Recursive Neural Networks also exist, these allow cycles to previous layers. . .
- ▶ Alphago was an RNN. Alphago zero is better and used a “two-headed” architecture:
 - ▶ A **value network** that attributes values to board positions
 - ▶ A **policy network** that links board positions to actions that realise them
 - ▶ It is essentially making a giant decision tree, which is pruned to a manageable set by assigning values to states without seeing them through to outcomes.
- ▶ This is all beyond the scope of the course, but you might wish to examine how these work

Auto encoders



- ▶ Auto encoders provide a low-dimensional representation of the data
- ▶ They consist of separable parts, the encoder and the decoder
- ▶ They can be used for de-noising
- ▶ They are particularly useful when data are limited

Summary

- ▶ Neural Networks are possibly the most important development in AI.
- ▶ They provide universal approximation, allowing non-parametric approaches to wide problem sets
- ▶ Network design is critical, and still very much an art
- ▶ If you understand the building blocks just a little, you can access others' networks and potentially tweak them

References (1)

- ▶ Chapter 11 of **The Elements of Statistical Learning: Data Mining, Inference, and Prediction** (Friedman, Hastie and Tibshirani).
- ▶ Russell and Norvig **Artificial Intelligence: A Modern Approach**
 - ▶ **Chapter 20 Section 5: Neural Networks**
- ▶ Swish: Ramachandran, Zoph and Le **Searching for Activation Functions**
- ▶ Important historical papers:
 - ▶ McCulloch and Pitts (1943) A logical calculus of the ideas immanent in nervous activity
 - ▶ Minsky and Papert 1969 Perceptrons
- ▶ Theoretical practicalities:
 - ▶ Practical advice from Bengio 2012 **Practical Recommendations for Gradient-Based Training of Deep Architectures**
 - ▶ Kull et al 2019 NeurIPS **Beyond temperature scaling: Obtaining well-calibrated multiclass probabilities with Dirichlet calibration**

References (2)

- ▶ Important historical papers:
 - ▶ Hecht-Nielsen, Robert. "Theory of the backpropagation neural network." Neural networks for perception. Academic Press, 1992. 65-93.
 - ▶ Bishop 1994 *Mixture Density Networks*
- ▶ Likelihood and modelling applications of Neural Networks:
 - ▶ Chilinski and Silva *Neural Likelihoods via Cumulative Distribution Functions*
 - ▶ Albawi, Mohammed and Al-Zawi *Understanding of a convolutional neural network*
 - ▶ Omi, Ueda and Aihara *Fully Neural Network based Model for General Temporal Point Processes*