Neural Nets and the Perceptron (Part 2, Deep Networks)

Daniel Lawson — University of Bristol

Lecture 09.1.2 (v1.0.2)

Signposting

- This Block is split into two Lectures:
  - 09.1 (this lecture) on the theory
  - 09.2 on practicalities
- Lecture 09.1 is further split into two parts:
  - Part 1: Introduction and the perceptron
  - Part 2: Multi-layer Networks
- This is Part 2, which covers:
  - Multi layer perceptron and the feed-forward neural network
  - Learning for deep neural networks
  - Other types of neural networks

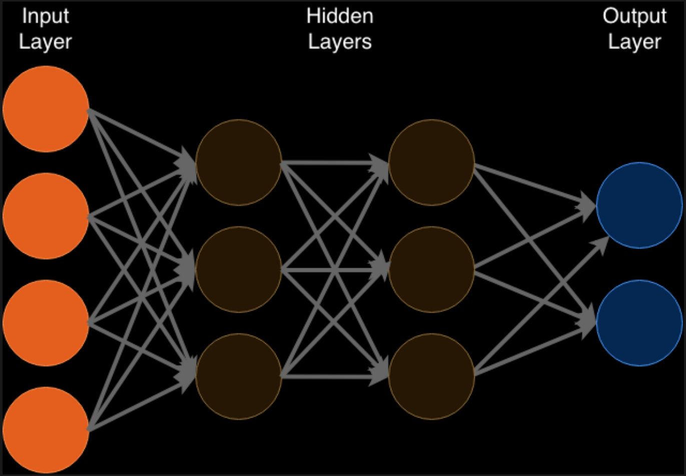
ILOs

- ILO2 Be able to use and apply basic machine learning tools
- ILO3 Be able to make and report appropriate inferences from the results of applying basic tools to data

Multilayer Perceptrons

- We have discussed the basics of how Neural Networks function
  - These had only **single layers**
- Most of what is important in Neural Networks comes from the addition of **hidden layers**
  - Hidden layers can be treated exactly as the layers we have observed
  - It is the mathematical tools that allow these to be used modularly that is transformative

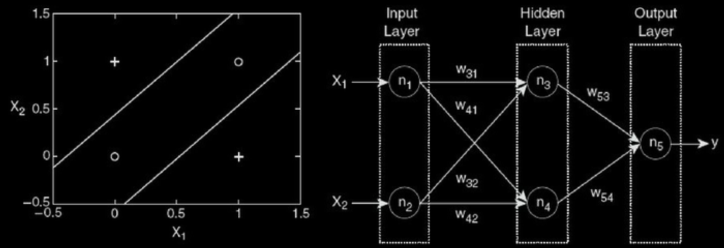
Multilayer Perceptrons / Feed Forward Neural Networks



Multilayer Perceptrons / Feed Forward Neural Networks

- **Architecture** choices include the number of layers and the connectedness
- Important issues include:
  - 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.

- Any[1] 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**

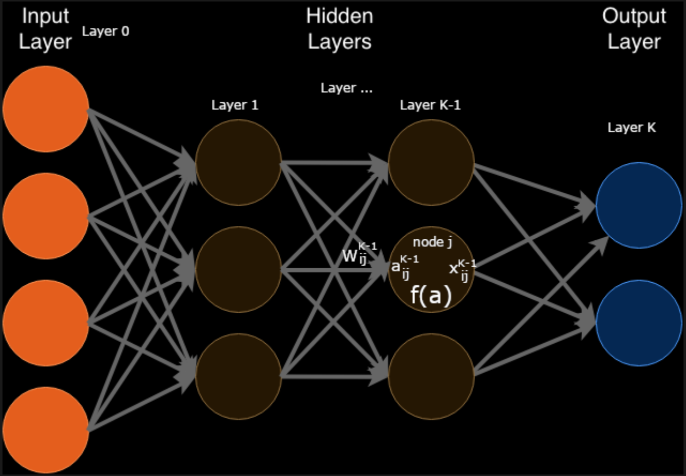
[1] continuous, compact function on $\mathbb{R}^n$

Back Propagation

- Learning Neural networks was an art until **back propagation** was discovered[2].
- 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...

[2] Hecht-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.

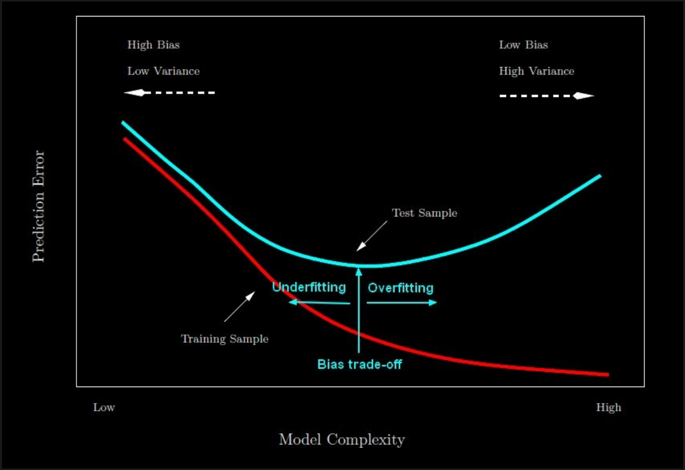
- **Stochastic Gradient Descent** addresses local minima and computational cost together.
  - It uses **mini-batches** of data for a gradient update.
  - This makes each update **random**, creating a type of **annealing** in the algorithm:
  - We can take large random steps when we are far from the optima (large step size),
  - And much shorter and hence on average reliable steps when we are closer (small step size).

Additional notes on learning

- Learning a Neural Network is still non-trivial. Start with this advice[3]
  - **Second order methods** are often used later in the fitting process, closer to the global optima.
  - **Hyperparameters** matter. Some optimisers, e.g. Adam, can tune them semi-automatically. Standard ones require **manual tuning** for e.g. step size.
- There is nothing here to prevent **overfitting**!

[3] Bengio 2012 Practical Recommendations for Gradient-Based Training of Deep Architectures

Learning rates



- **not** specific to neural networks
- But particularly important due to NN flexibility

Hints on overfitting

- Many optimizers include options for these tricks and more:
- **Penalize** large weights:
  - Ridge (L2) penalisation: $L = L_0 + \lambda \sum_{i,j} |W_{ij}|^2$
  - Lasso (L1) penalisation: $L = L_0 + \lambda \sum_{i,j} |W_{ij}|$
- **Dropout**:
  - New hyperparameter p_k for layer k: the **dropout rate**
  - Each learning step, with independently randomly set all outputs from a neuron to 0
- **Early stopping**:
  - retain a test dataset (from the training dataset)
  - evaluate performance on the held-out set
  - stop when this no longer increases

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 Networks[4]
  - Calibrate probabilities in a "post processing" layer[5]
- 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**.

[4] Bishop 1994 Mixture Density Networks

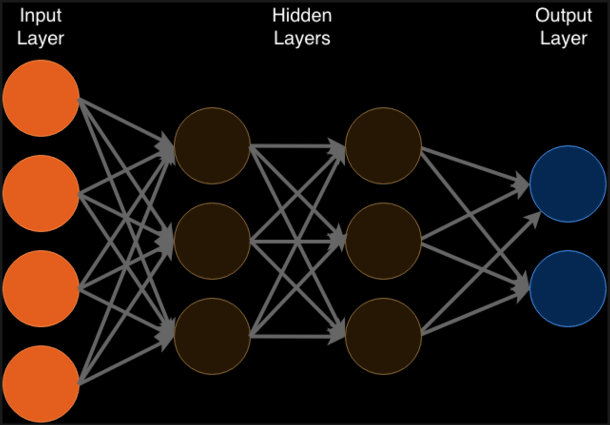
[5] Kull et al 2019 NeurIPS Beyond temperature scaling: Obtaining well-calibrated multiclass probabilities with Dirichlet calibration

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.



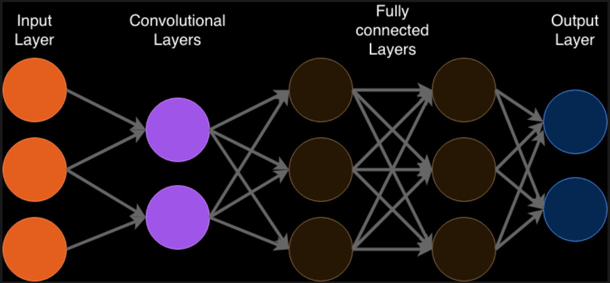
Feed forward neural network

- 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 better[6].

[6] Chilinski and Silva Neural Likelihoods via Cumulative Distribution Functions

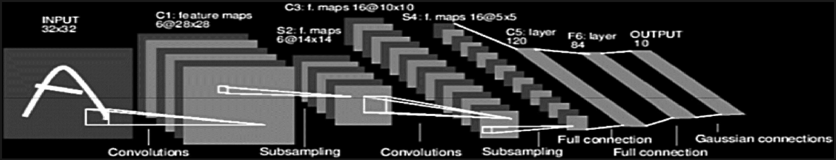
Convolutional neural network

- 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



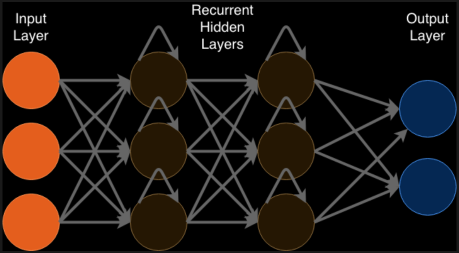
- CNNs are a core part of image processing[7]
- They scan an image, constructing **features**
- Different convolutions can create different features, including:
  - Larger objects
  - Edges
  - Presence/absence of either via max-pooling

---

[7] Albawi, Mohammed and Al-Zawi Understanding of a convolutional neural network

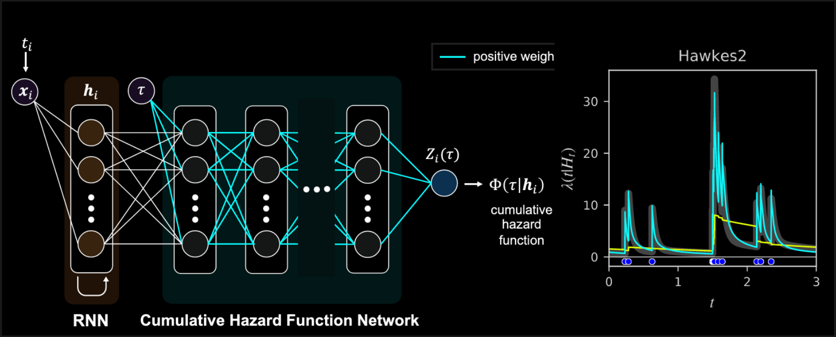
Recurrent Neural Network

- 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



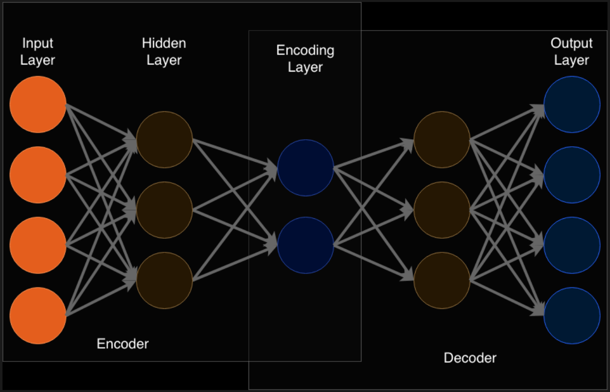
- An RNN acts as a "memory" for an arbitrary history[8]
- A CNN acts as a universal approximator to the CDF
- This is translated into the Likelihood of the data by back-propagation differentiation

[8]Omi, Ueda and Aihara Fully Neural Network based Model for GeneralTemporal 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

Reflection

- What advantages and disadvantages do Deep Neural Networks present?
- How straightforward are they to apply? Under which circumstances?
- Why are they not more used as a universal approximator?
- By the end of the course, you should:
  - Understand a neural network at a basic level
  - Be able to appropriately select deep learning methods and architecture
  - Be able to work with the mathematics underpinning perceptrons

Signposting

- Still to come:
  - Lecture on the practicalities of Neural Networks
  - Workshop on using them in practice

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
- 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 GeneralTemporal Point Processes