Neural Network Architecture and Practicalities

Daniel Lawson — University of Bristol

Lecture 07.2 (v2.1.0)

Questions

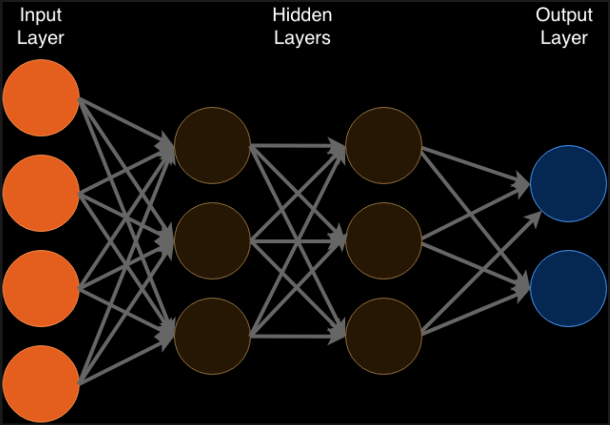
- What are the most important types of neural net?
- What role does architecture have?

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.



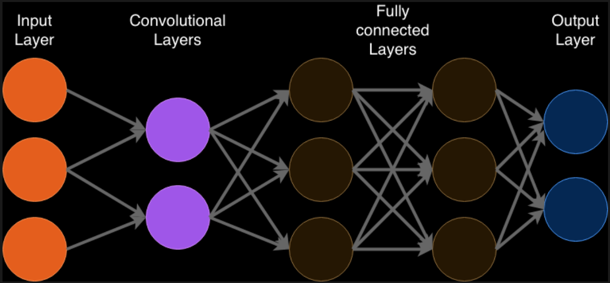
Feed forward neural network

- 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[1].

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

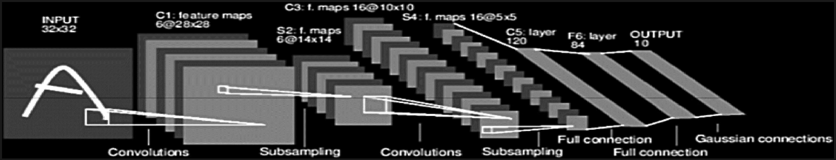
Convolutional neural network

- 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

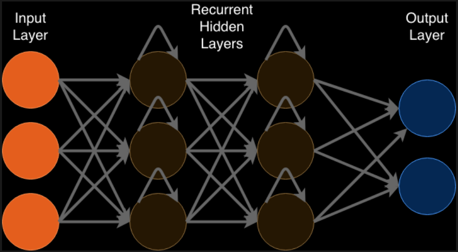


- CNNs are a core part of image processing[2]
- They scan an image, constructing **features**
- Different convolutions can create different features, including:
  - Larger objects
  - Edges
  - Presence/absence of either via max-pooling

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

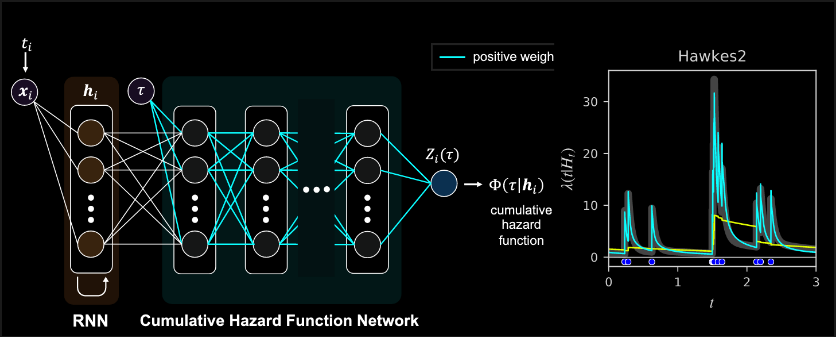
Recurrent Neural Network

- 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



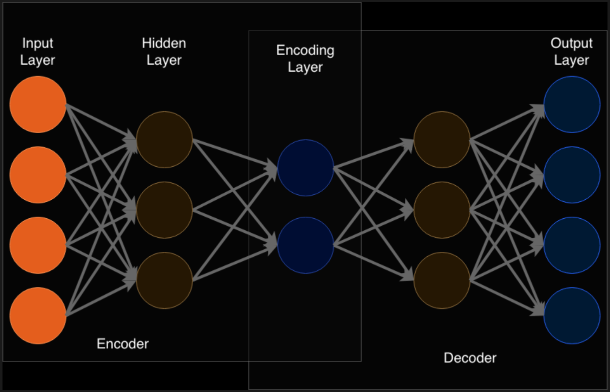
- An RNN acts as a "memory" for an arbitrary history[3]
- A CNN acts as a universal approximator to the CDF
- This is translated into the Likelihood of the data by back-propagation differentiation

[3] Omi, 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

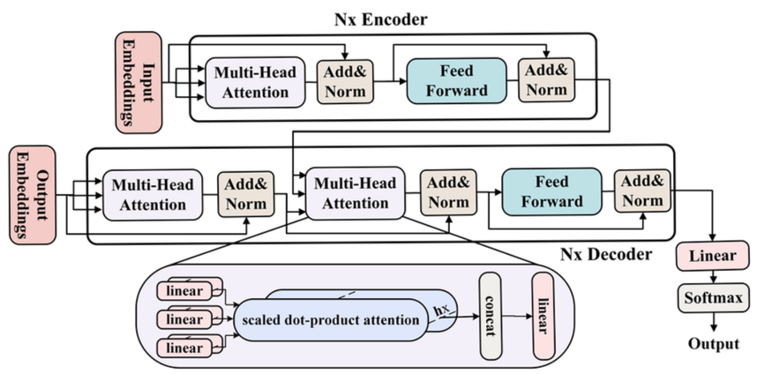


- Auto encoders provide a low-dimensional representation of the data
- They consist of separable parts, the encoder and the decoder
- They can be used for de-noising
- They are particularly useful when data are limited

LLMs, Foundation Models

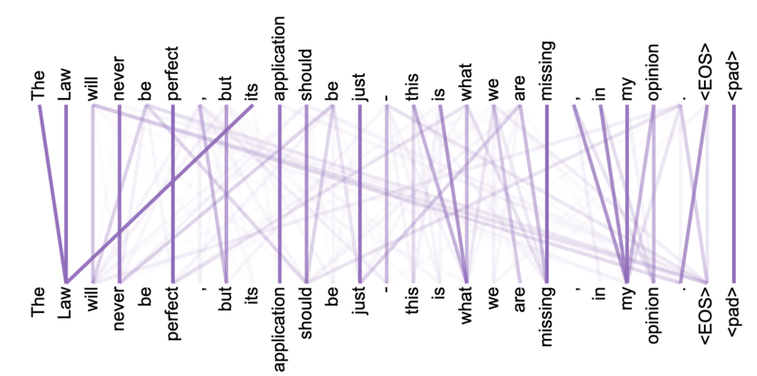
- A Large Language Model (LLM) is the basis of modern Chatbots
- They are essentially very large transformers
- Trained on very large datasets
- To predict the next ‘token’
- For a very long time!
- Attention is parallelizable learning, e.g. `the fat cat sat on the mat`
  - learns `(the fat)` → `cat` and `(the fat cat)` → `sat` simultaneously
  - solves the **vanishing gradient problem**, keeping context over long distances
- Foundation models are trained on more data types

Transformers



doi:10.48550/arXiv.2303.12914

Attention



From ‘Attention is all you need’ (2017) by A Vaswani · Cited by 137673

Finetuning

- In practice, **good neural networks are very large**
- Many problems contain very similar structure
- In practice you therefore want to **download someone elses' model**
- You then **finetune** it to your task
- Best practice learns a 'low rank' approximation, e.g. LoRA (`https://github.com/microsoft/LoRA`)

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

Implementing Neural Networks

- Implementations are best though of in two classes.
- **Simple networks** have a restricted architecture and can be deployed "out of the box" as a Machine Learning tool.
  - Examples include sklearn.linear_model.Perceptron, R's neuralnet packages, etc
  - Often either shallow or very simple hidden layer structure
- **Deep networks** require a complex specification of architecture and significant computational optimisation, so are very large (and mercifully, open source) endeavours
  - This is the focus here.

Deep NN Implementations

- There are two main libraries for deep neural networks:
- **TensorFlow**, developed by Google Brain.
  - Well documented
  - Easier to use
  - Industry standard
  - Tensorboard visualisation is useful
- **PyTorch**, developed by Facebook.
  - Newer, less support
  - Dynamical coding paradigm: graph can remodel in the light of the data
  - Debugging is easier? As the code is compiled at runtime, like native python

Using implementations

- Tensorflow is a low-level language. You can interact with it through abstraction layers which allows very simple implementations.
  - Keras is very widely used and makes accessing TensorFlow very easy.
  - PyTorch is already conceptually a “high level” implementation.
- Keras can use various **backends** (implementations):
  - TensorFlow
  - MXNet
  - Theano is a pure python library for a wide class of array computation, not just Neural Networks. It was forked into Aesara...
  - Microsoft Cognitive Toolkit, but this is no longer in active development.
- See Tensorflow or keras?

Practical advice

- Explore recommendations. e.g. Practical Advice for Building Deep Neural Networks:
- As a starting point:
  - Use the “adam” optimizer
  - Use a ReLU activation function
  - Remember not to use an activation function for the output layer (except for classification, when use a sigmoid)
  - Add bias to every layer (shouldn’t have to worry about this in keras)
  - Whiten (normalize) your input data (we’ll see this in the workshop)
- **Don’t believe me.** Get other opinions, and try things yourself.

Debugging

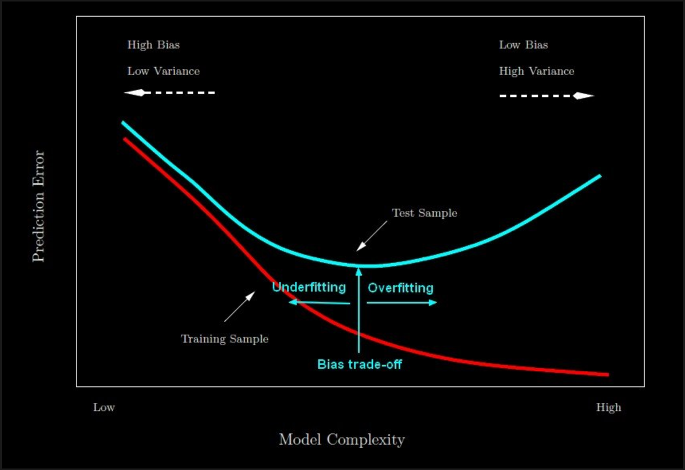
- Check the **input data**. . .
- For many tasks:
  - **OVERFIT**. "Accuracy should be essentially 100% or 99.99%". If it isn't, the network isn't flexible enough, or learning correctly.
- Change the learning rate
- Decrease mini-batch size
- Remove batch normalization (this exposes NA values)
- Reconsider the architecture
- PLOT your results! training loss by epoch is a natural plot

Additional notes on learning

- Learning a Neural Network is still non-trivial. Start with this advice[4]
  - **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**!

[4] 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

Further reading

- Keras and PyTorch
- Tensorflow or keras?
- A performance focussed comparison: TensorFlow, PyTorch or MXNet?
- Tensorboard
- Brilliant.org on Backpropagation
- Practical Advice for Building Deep Neural Networks