Machine learning workshop: modern deep learning

Alex Richardson

February 23, 2024

Structure

- Some practical concerns
- Different kinds of training
- Neural network architecture zoo
- Neural differential equations
- Advanced jupyter notebook examples

What is deep learning?

- **Deep learning** is just machine learning on deep neural networks
    - Deep generally means more than 2 layers

What about Artificial Intelligence?

- Artificial Intelligence (AI) is a common phrase that is often misused or ill defined
    - Especially with popular science, journalism and funding bodies
- Broadly it is the study of intelligence in machines or software
- From about 2012 onwards, deep learning techniques have produced the best results in AI research

Neural networks are interpolators

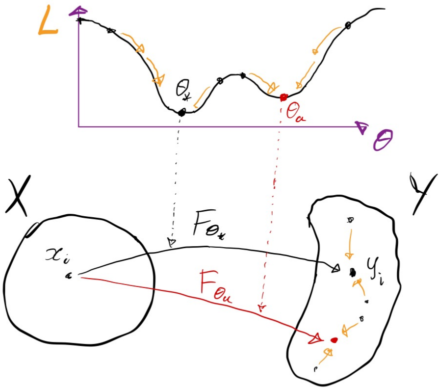
- The network we trained on the MNIST dataset could perform well on the training dataset, but does it **generalise** well to the test dataset?
- Often models **interpolate** the training data very well, but struggle to **extrapolate** to new unseen data
- This is called **overfitting**, and can be minimised with a few tricks:
  - Get more data!
  - Regularise the parameters (i.e. add a term to the loss function that is minimised by small parameters, leading to smoother F)
  - Train for less time
  - Choosing an appropriate model structure that explicitly suits the data (referred to as an **inductive bias**)

Understand your loss function

- If all is well and your loss function is minimised, what actually happened?
- Does your loss function being minimised guarantee a good model?
    - This is more important for more complex models, for example large language models, or AI that makes decisions that affect people...

Local minima

- We are not guaranteed to find the global minimum, just a local minimum



Always start simple and build up

- Throwing the biggest fanciest neural network at your problem might work, but you're unlikely to understand why, and it will probably be computationally expensive
- Instead you should try the simplest possible models, and when they don't work, gradually increase their complexity until they do
- That way you will:
  - Understand the bit that helped make your model work
  - Not be wasteful in computation

A brief note on ethics...

- A lot of current machine learning research is done by large companies (Google, Meta, Microsoft...)
  - There is therefor a profit incentive
- A lot of recent success in ML (i.e. chat-gpt) is down to huge amounts of processing power and huge volumes of training data
  - Daily power use of ChatGPT queries is approximately the daily power usage of 30000 average US homes
  - Sometimes data harvesting techniques are unethical / illegal
- Always think about what your model *could* do for someone else
  - Image segmentation or recognition is all fun and games until the police / military use it

A stronger point about ethics

- Some AI researchers hype up the dangers or risks of general artificial intelligence
  - This is when a sufficiently complex computer system becomes as intelligent and self aware as humans
- This is largely marketing, and distracts from the dangers of our current usage of machine learning based tools
  - If you hype something up as dangerous or risky, that makes it sound really cool and powerful to some people

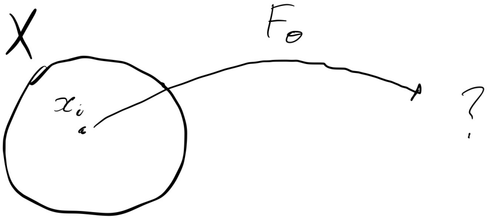
Beyond classification

- The examples we have seen here so far are classification
  - Input set X is some data (images)
  - Output set are labels (integers 0-9)
- There are many other ways to set up a machine learning problem, and some of them are quite interesting

Unsupervised vs Supervised

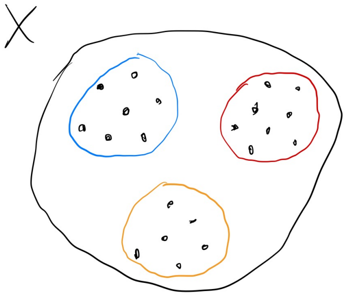
- Supervised learning is when we have sets X and Y. This includes
  - Classification - predicting labels
  - Regression - predicting numbers
- Unsupervised learning is when we only have X. What can we learn about X without Y?

Unsupervised learning



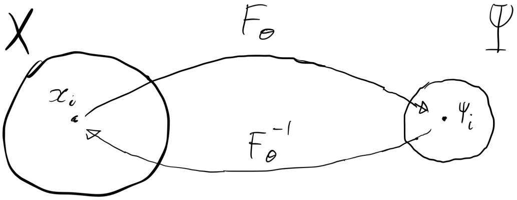
Unsupervised learning - clustering

- One nice idea is to try and organise each x_i into clusters, based on how they are arranged in X



Unsupervised learning - latent space

- We can construct an artificial output space, commonly referred to as a **latent space** Ψ, and learn mappings to **and from** Ψ
- By constraining or defining Ψ appropriately, we can learn something about X

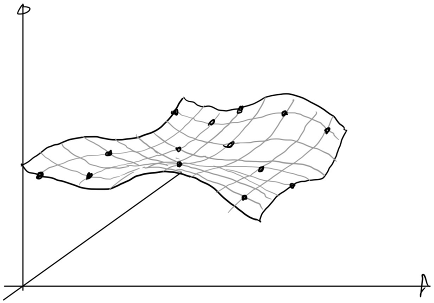


Unsupervised learning - latent space

- Usually a latent space has lower dimensionality than X
    - Often this **encoding** to a latent space is a first step / trick that's used before other machine learning techniques
    - It's cheaper to train models on a lower dimensional latent space
- A linear approach to construct a lower dimensional latent space is by Principle Component Analysis (PCA), where we project onto the m largest eigenvectors of the covariance matrix of the dataset X
- A common neural network that learns latent space mappings is an **Autoencoder**

Unsupervised learning - manifold learning

- We can go a step further in the maths - by assuming the data in X is actually constrained to an embedded manifold of much lower dimensionality
    - If true, learning a projection to this manifold would be very useful in understanding X



Unsupervised learning - generative models

- We can also assume that x_i in X are sampled from some probability distribution $p(x_i)$
  - If we learn $p(x_i)$, we can then **generate** new samples from p
  - Naively we can assume a normal distribution with mean and variance defined from X, but there are many fancier ways to parameterise p

Transfer learning

- Sometimes a network trained on one task can be re-used on another task
    - Sometimes *parts* of a trained network can be used in other networks
- Transfer learning is the process of taking a pre-trained model from one task and modifying it to work on another

Summary so far

- Training neural networks is hard
  - The parameters are high dimensional
  - The loss often get stuck in local minima, but sometimes that's ok
  - It's worth carefully considering how much we trust a trained model
- Training is versatile
  - There are many different ways to define a function we're trying to train
  - There are many different ways to constrain that function, even with a limited or incomplete dataset

Fully connected feed-forward neural networks

- This basic neural network structure is easy to define, but scales badly with large datasets
    - In *principle*, from the universal approximation theorem, this basic neural network can do anything
    - In practice, when we have the very large space of parameters Θ that are required, optimising for an optimal θ_* is very expensive
- Time to exploit symmetry!
- The structure and arrangement of layers or parts of a neural network is referred to as the **architecture**
    - This is basically everything that's not a trainable parameter

Convolutional Neural Networks (CNNs)

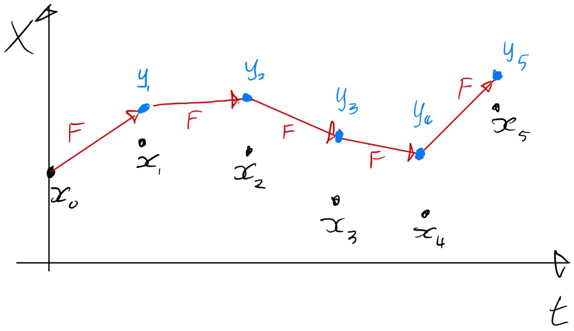
- Here we exploit local spatial structure in our data
- If inputs are images, we can assume that nearby pixels have more important interactions than distant ones
- Just learn the weights of a discrete convolution kernel
- `https://github.com/vdumoulin/conv_arithmetic/blob/af6f818b0bb396c26da79899554682a8a499101d/README.md`

Recurrent Neural Networks (RNNs)

- Often we have a network that maps between spaces of equal dimensionality: $F : \mathbb{R}^n \to \mathbb{R}^n$
- Recurrent neural networks are trained to be applied several times in series: $F(F(F(\dots F(X))))$
    - If F has ℓ layers, and we repeat it R times, this is similar to a single feed-forward neural network of $\ell \times R$ layers, but with many less parameters to store/train

Recurrent Neural Networks (RNNs)

- RNNs work well for time series data - each application of F can be trained to move forward through time



Residual Neural Networks (ResNets)

- In the simple example earlier, a layer (combining the nonlinear and linear parts into one) of our neural network is represented as:
$$h_{i+1} = \sigma_i(L_i(h_i)) = g_i(h_i)$$
- A residual network is just one where we have this form:
$$h_{i+1} = h_i + \sigma_i(L_i(h_i)) = h_i + g_i(h_i)$$
- We are now learning a **residual** increment to each hidden layer, rather than a complete transformation

Autoencoders

- Autoencoders learn an approximation of the identity function, but where the middle layers are much smaller than the input/output
- A successfully trained autoencoder learns to **encode** data to a lower dimensional **latent space**, and then **decode** it back to X
    - This is like learning a compression algorithm that's dependant on data

Graph neural networks (GNNs)

- If our data X is represented by a graph (network), then Graph Neural Networks (GNNs) are a natural choice
- Generalisation of the idea of a CNN
  - Learn parameters to combine local (along edges of the graph) information into a new graph at each layer
  - CNNs are just GNNs but on a square lattice graph

Transformers and Large Language Models (LLMs)

- Transformers are the popular method for large language models (LLMs), the T in Chat-GPT
- They are trained to predict the next word in a string of text/sentence
- First each word of a sentence is represented as a vector via a pre-trained **word embedding**
- Transformers use a **attention** mechanism:
  - Learn some matrices to transform the word embedding vectors
  - These transformed word embeddings are multiplied together to give an 'importance' of each word to each other word
  - These word importances are averaged together, and fed through some fully connected layers to produce a probability distribution of the next word

Transformers as autoregressive models

- If you start with a sentence, and predict the next word, you have a new longer sentence
- This can be fed back into the input, creating another slightly longer sentence
- Including this behaviour in the training gives an **autoregressive** model
- This is the basic idea of Chat-GPT (and other LLMs) - they are just models that predict the next word given a body of text as input
- This is how Chat-GPT can generate large volumes of text

Physics Informed Neural Networks (PINNs)

- Technically more of a training technique, but you can augment the loss function with a term that describes a differential equation you expect your data to satisfy
- Popular in where we have some underlying conservation laws (i.e. momentum conservation), but don't know the full dynamics or structure of the data

$$L(x, y) = \|F_\theta(x) - y\| + \|\nabla F_\theta(x) + \partial_t F_\theta(x)\|$$

Combining things

- There's sometimes no clean distinction between different neural network architectures
- It makes sense to combine many together:
  - Recurrent Convolutional Neural Network for data that has both spatial and time sequence structure (videos)
  - Recurrent Residual Neural Networks for learning repeated incremental updates to data
  - Convolutional networks on latent spaces
  - Physics informed graph neural networks
- Combining things together has some space for creativity

Summary

- There are many different architectures of neural networks
    - Choosing the right one for your task makes a huge difference
- A lot of machine learning research is focused on creating / finding new architectures that outperform previous models on given tasks

Opinions on future research directions

- Finding new architectures is appealing, but can be quite ad hoc
    - Often more engineering than science
- Often the actual optimisation algorithm is as important as architecture, so research effort spent there is good too
- We have a lot of stuff that works, but little understanding of *why*
    - We need more theory work

Tidying things up a bit

- This is all a bit messy - there are so many different names and types of network
- Is there some nice unifying structure behind it?
  - Yes - neural differential equations
  - RNNs and CNNs can be reframed as discretisations of different neural differential equations

Discretisation

- Consider a recurrent ResNet:

$$y_{i+1} = y_i + F_\theta(y_i)$$

- This can be re-interpreted as an Euler's method (first order taylor series) discretisation of a neural ODE:

$$\frac{dy}{dt} = F_\theta(y(t), t)$$

Neural Differential Equations

- These are just differential equations with a neural network in them:
$$\frac{dy}{dt} = F_\theta(y(t), t)$$

- We can think of them as a continuous / infinite depth generalisation of RNNs
- This also allows us to use the mathematical tools for differential equations in machine learning
- We typically solve for y:
$$y(T) = y(0) + \int_0^T F_\theta(y(t), t)dt$$

Neural Differential Equations

- CNNs can be considered as a discretisation of a parabolic PDE
- RNNs can be considered as a discretisation of an ODE
- We can use neural networks in many different differential equation structures
  - Hamiltonian dynamics
  - Stochastic Differential Equations (SDEs)
  - Control Differential Equations (CDEs)
- This can help come up with new neural network architectures
- This can also help actually understand a trained neural network

Neural Differential Equations

- As an example from my own research, I've been looking at a neural network parameterisation of reaction-diffusion-advection PDEs:

$$\frac{\partial c}{\partial t} = D_\theta \nabla^2 c + \nabla \cdot (U_\theta(c) c) + R_\theta(c)$$

- Here R_θ and U_θ are different neural networks (and D_θ is a constant)
- When this equation is discretised and solved, we get a complex recurrent convolutional architecture
- When trained, we can still interpret D_θ, U_θ and R_θ in terms of biomechanical processes

Generative diffusion models as neural SDEs

- A popular method for generating images is with **diffusion models**
- The basic idea is to train an image denoising function, and repeatedly run it on pure noise to generate a new image
- This can be nicely formulated as a stochastic differential equation

Generative diffusion models as neural SDEs

- Define the forward process as:
$$dx = f(x,t)dt + g(t)dw$$
- If we start with $x(0)$ as an image, and solve for $x(T)$, we get an increasingly noisier image
- By defining f and g correctly (usually $f = 0$), we can guarantee convergence to a normal distribution at some time T
- We can define the reverse SDE:
$$dx = [f(x,t) - g^2(t)\underbrace{\nabla_x p_t(x)}_{\text{learn this}}]dt + g(t)dw$$
- Where $p_t(x)$ is the distribution of $x(t)$

Generative diffusion models as neural SDEs

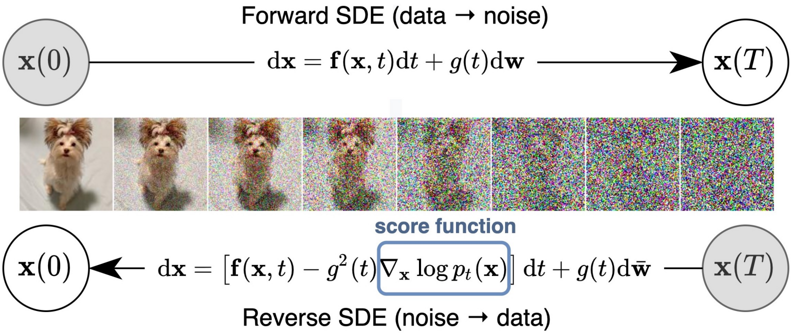

Forward SDE (data → noise)
$\mathbf{x}(0)$
$\mathrm{d}\mathbf{x} = \mathbf{f}(\mathbf{x}, t)\mathrm{d}t + g(t)\mathrm{d}\mathbf{w}$
$\mathbf{x}(T)$
score function
$\mathbf{x}(0)$
$\mathrm{d}\mathbf{x} = \left[\mathbf{f}(\mathbf{x}, t) - g^2(t)\boxed{\nabla_{\mathbf{x}} \log p_t(\mathbf{x})}\right] \mathrm{d}t + g(t)\mathrm{d}\bar{\mathbf{w}}$
$\mathbf{x}(T)$
Reverse SDE (noise → data)

Figure: `http://yang-song.net/blog/2021/score/`

Generative diffusion models as neural SDEs

- By learning $s_\theta = \nabla_x p_t(x)$ we can then take random noise from a normal distribution, and iteratively denoise to generate an image
- If s_θ also depends on some class or label associated with the data it's trained on, we can control what we're sampling
- For text to image generators, we add a word embedding part to s_θ and train on image-text pairs

Final notebooks

- There are two jupyter notebooks exploring more advanced ideas. Work through whichever you find interesting:
    - Autoencoders on MNIST, combining different models together, augmenting a neural network with fixed functions (fourier transforms), transfer learning
    - Neural ODE that ‘counts’ by mapping digits in MNIST to the next one along