NOTES, DON'T PRESENT THESE!

Mark Blyth

What and why?

- ML is ‘fancy’ model fitting
  - We seek some model that we can use to assign meaningful output values
  - Goal is to create functions that are general enough to fit to any data
- ODE solutions could be a rich source of models to fit
  - Timesteps in the ODE solution are like layers in an NN
  - Adaptive ODE solvers would allow us to propagate info through the NN, while guarding error
  - Lots of research on ODEs already, so we have a good basis for existence, uniqueness, etc.

Section 1: forumation

- Let $z(T, x)$ be a solution to an ODE, evaluated at time T, initial condition x
  - Can be found easily, by numerically integrating the chosen ODE
  - Even simple nonlinear ODEs can have very complex solutions
  - Typically, can't be expressed in terms of elementary basis functions
  - Derivative operator can therefore be thought of as an easy way to get 'richness' out of simple functions
  - This richness might be rich enough to let us do ML
  - How can we 'tune' our ODE so that z is in some way useful?
- $\frac{\mathrm{d}z}{\mathrm{d}t} = f(A(t), z), \;\; z(0) = x$
  - Add some controller $A(t)$ to make the solution do something useful
- $u(x) = \mathbf{a} \cdot \mathbf{z} + b$
  - Define a scalar OBSERVATION from the flow map
- $\mathrm{argmin}_{\mathbf{a},b,A} \sum (y_i - u(x_i))^2$
  - Training is then the observations and controllers that minimise the square-error
  - This is an optimal control problem!

Finding a controller

- $\text{performance} = \int \text{error}^2 \mathrm{d}\mu(x)$
  - Start off with an error metric, given by the integral over all possible errors, weighted by some probability measure
- $\frac{\mathrm{d}\ \text{performance}}{\mathrm{d}A} = \int \frac{\mathrm{d}\ \text{error}}{\mathrm{d}A} \mathrm{d}\mu(x)$
  - Chain-rule to find the gradient of the performance w.r.t. controller
- $\frac{\mathrm{d}\ \text{perturbation}}{\mathrm{d}\tau} = J_z \text{perturbation}$
  - We linearise about some small (controller-induced) perturbation, to get the linear variational equations for the perturbation dynamics
- $\frac{\mathrm{d}\ \text{output}}{\mathrm{d}A} = \text{perturbation}(T)$
  - Effect of some small controller perturbation is approximated by the leading-order perturbation size at the end of the flow-map time
- We can then use gradient descent or something to optimize the controller, so that the flow map is useful to our ML problem

Connection to DNN

- Deep NNs are a dynamical system that can change dimension
  - Each 'layer' (function / neuron output) feeds into the next (function / neuron input)
  - Input gets linearly transformed, then a component-wise nonlinearity is applied
  - Dynamical system!
  - Can change the dimensionality by using non-square linear transforms
- Continuous NNs cannot change dimensions
  - Doesn't make physical sense for the dimension of an ODE flow to change as a function of time
  - We must either project into a higher-dimensional space at the start of the flow map, at the end, or not at all
  - At the end doesn't really make sense since we're already projecting down onto feature space; nothing to gain by an additional projection
- Continuous NNs can overcome issues with training deep NNs
  - Training DNN is hard, as gradients can explode or vanish, causing gradient descent to stop working
  - Imposing structure on the ODEs, such as Hamiltonian structure, could help ensure gradients remain 'useful', and overcome these issues
  - We can use existing numerial methods, like adaptive timestepping solvers, to solve for long time-horizons, necessary when 'lots of computation' is needed

Connection to resnets

- Residual neural networks overcome vanishing gradients by selectively omitting layers
  - Neuron values are small, so gradients end up being the product of lots of small numbers, and quickly vanish to zero as we add more layers
  - Residual neural networks selectively learn to skip layers, which reduces the vanishing gradient problem; this is effectively like the NN adaptively learning its own architecture, and therefore has a lot in common with ODE solvers adaptively choosing step sizes
- The dynamical systems viewpoint explains why this should help training
  - Not particularly interesting from our perspective; basically the NN learns to identity-map some layers, which DS perspective shows would make sense
- Resnets learn an Euler-discretisation of an ODE
  - The learned identity maps mean that output = input + perturbation
  - This is equivalent to solving an ODE with the forward Euler method
  - Adaptively choosing the stepsize, as ODE solvers tend to do, is equivalent to adaptively changing the layers to minimise output error

Representability and controllability

- We need to be sure that the flow map can process data as desired
  - Need to be sure that, given some flow map and some training data, we are able to find a suitable control strategy that'll let us do something useful (classification, regression) with the data
- This is a problem of controllability
  - Controllability: given some initial condition and some target, can we drive the system to a target neighbourhood in finite time?
  - ML version: we apply some post-processing step to the flow map output, eg. linear regression. Given this final (supervised) learning model, can we control the flowmap to provide satisfactory accuracy on this learning model?
- Idealised problem: can the flow-map model arbitrary mappings on the data?
  - Defines a multiplicative control (not particularly clear why they would do this), and shows that the control should be state-independent; this hugely limits the predictive power
  - Slightly contrived, but nicely demonstrates that exact representation is hard. Instead, we should ask how well can we approximate.
- What happens if we use a kernel method?
  - 'Boost' dimensionality, so that the system has more DoF
  - If we can smoothly transform our target map map to the identity map (no longer arbitrary!), then we can find an ODE whose flow-map can model the

Continuum in space

- PDE models are useful when we have spatially structured data
  - Images have a spatial structure, audio can be translated into a spectrogram for easier processing, which produces a 2d image
  - We can model this structure with PDEs
  - Actually though, that's not necessarily a good model; we want non-local dependence, eg. we want to extract information from, say, edges or curves, rather than purely locally
- Using a convolutional kernel gives CNN-like behaviours
  - We can model non-local dependence using the convolutional kernel
  - In this case the spatial models start to act like convolutional neural networks again

Constraints, structure, and regularisation

- We can add constraints to the system
  - Eg. have an orthogonal control matrix
  - No explanation as to why we would want to or what this would achieve
- We could add structure
  - Already discussed; could use Hamiltonian structure to help prevent vanishing gradients
- We could add regularisation terms
  - Could limit the total control action, or some norm thereof
  - No explanation as to why this would be useful or interesting
  - Presumably since it's a numerical system, there's no penalty for having large control energy