Neural Networks for Dynamical Systems

AMATH563: Homework 4

Dan Gorringe
dngrrng@uw.edu

June 14, 2020

Sec. I. Introduction and Overview

Neural networks are used to model a series of dynamical systems. It's found that this approach is compute intensive, and that parameter-tuning is difficult. The theoretical underpinnings are beautiful, but it seems a GPU, IPU, or TPU might be beneficial in this quest.

The findings come from a Python Notebook that can be found on my Github[1].

Sec. II. Theoretical Background

Deep Learning

A giant in the world of machine learning buzzwords, Deep learning, is simply a neural network with many hidden layers. Neural networks are, simply put, a system of matrix multiplies with element-wise functions pre-applied. Many architectures exist, some sequential, some recurrent, others fully connected, some increasing dimensions, and others decreasing - The neural network zoo is vast, and equally exciting.

The explosion in deep learning has been catalysed by access to vast data, and innovations in computation. There is huge applicable utility, especially in the domain of high dimensional problems.

Activation functions

The default layer can be modelled as a matrix multiply. This inherently imposes a linear relationship, using activation functions non-linearity can be introduced. These are also known as hidden units.

$$\underline{y} = \underline{f}(\underline{\underline{A}}, \underline{x})$$

Activation functions are cleverly chosen in order for easy differentiation, to aid model weight learning. Common activation functions include; `linear`, `sigmoid`, `tanh`, and `rectified linear unit (ReLU)`.

Backpropagation

Updating model weights uses backpropogation, the differentiation chain rule, to progress to a minima of your loss function. Loss functions quantify the model objective, they are often chosen to make finding derivatives easy. Backprogation is the applciation of the the differentiation chain rule to find the derivative of the loss with respect to layer weights. Gradient descent uses backpropogation to approach a minima.

Stochastic Gradient Descent

The beauty in previous homework's has been illustrating the power in the low-dimensional nature of problems, and the efficiency of sparsity. Normal gradient descrent methods uses all data to find the gradient, while stochastic gradient uses a stochastic subset. The benefit is two-fold: a big reduction in compute, and the ability to escape troubling local minima.

Sec III. Algorithm Implementation and Development

KERAS

Tensorflow with KERAS abstracts all the difficulties in creating and training neural networks. Throughout a `sequential` neural network was used. The default hyperparamter values were mainly kept to, only changing learning rate once. Deep learning is hugely popular, the algorithm implementation and development is well abstracted and packaged.

Sec IV. Computational Results

Question 1: Kuramoto-Sivashinsky

A model was created with 5 layers. The first and last layers are both dense with no activation function. The middle three have twice the number of neurons and use `linear`, `sigmoid`, and `tanh` activation functions. Mean squared loss is used for 1024 epochs with a fifth of datapoints used as validation. The results found in Figure 2 seem to produce somewhat oscillatory characteristic behaviour modelled from Figure 1, the real data. The Kuramoto-Sivashinsky equation displays chaos - my intuition tells me that this should mean that the relatively small network used is insufficient to find generalisable results. Alternatively with more time and motivation more epochs might prove useful - but risk overfitting the one specific initial condition run used.

[1]https://github.com/DanGorringe/AMATH563_inferringComplexSystems

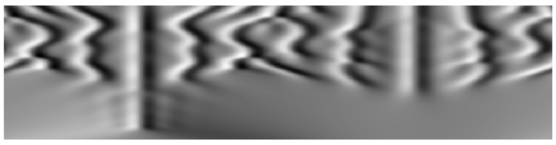

Figure 1: The sample data, from the `MATLAB` code provdied.

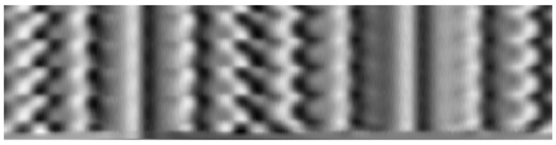

Figure 2: The prediction from initial conditions, from the trained model.

Running for new initial conditions in Figures 3 and 4 we see that fears of overfitting are well founded. The model needs to be refined in order to make generalisable predictions rather than superficial average pattern production.

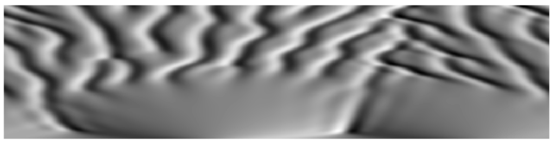

Figure 3: The real progression from the alternative initial conditions

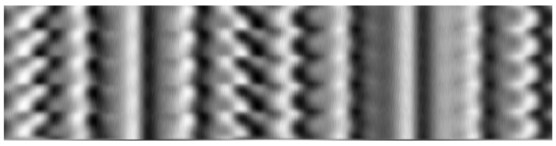

Figure 4: The prediction from alternative initial conditions, from the trained model.

Question 2: Reaction-diffusion

The reaction-diffusion equations given seem to be roughly analogous to a 2d Lotka-Voltera predator-prey system evolving over time. From the data produced from the program provided swirls rotating through time are produced.

By virtue of it's many dimensions the system needs to be lowered for efficient computation. SVD is used per instruction, and from the eigenvalues it would seem the system can be represented in less than 10 modes, see Figure 5.

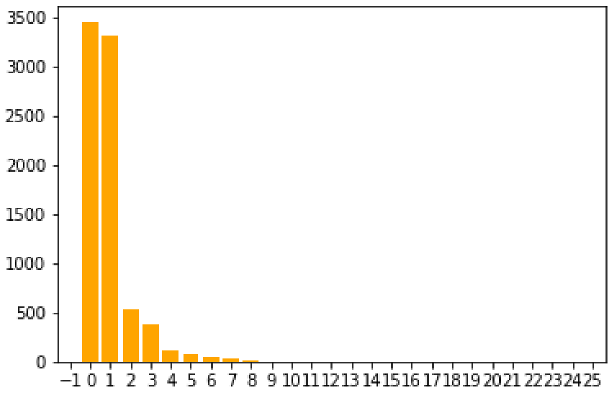

Figure 5: Obligatory chart of mode eigenvalues

SVD is used on both of the 2D timeseries. Using low dimensions as input and outputs to the neural network does not manage brilliant results. Qualitativly looking at the resulting GIFs from using low dimensions - poor recreation is found. The jumpy nature of low dimension predictions is assumed to be the fault of low dimension approximation. Using all 200 modes produces similar results to the original - notably a large reduction in comparison to the total pixel count. However given the mode eigenvalue values the image is expected to be of much lower dimension than is empirically found. SVD provides some reduction in compute cost, but not faithfully.

Looking at Figure 6 sheds some light on this issue. The data provided accounts for a small rotation, so the highest weighted mode is the average swirl position with it's lower counterparts accounting for minimal rotation when composed.

It is assumed that SVD performs badly because the modes produced are representative of the tiny temporal movement observed and not the generalisable movement implied. Given more data wildly different modes are to be expected, a small number of rotational modes should be able to compose the behaviour seen. SVD dimension reduction is unable to reach it's full potential as is limited by an unfair unreprestitive training set.

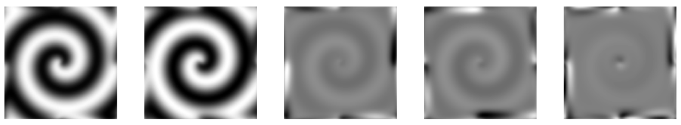

Figure 6: The top five highest energy SVD modes of training data.

This task highlights the difficulty with deep learning. With so many parameters it is difficult to create meaningful computationally-reasonable tests to compare hypotheses. Given no specialised hardware acceleration for machine learning; measuring metrics across learning rates, SVD truncation, and model architectures becomes rather hasslesome. This report's conclusions would benefit from in-depth time-consuming iterative statistic collection - though this problem of hyperparamter tuning is seen to characterise modern deep learning as a subject too.

Question 3: Lorenz

Uniquely for this question the data is generated in the Python Notebook used for the rest of the analysis - this provides a heightened degree of granularity or awareness of such that is in-often available. Having the power to collect a series of data means that one can ensure they truly match the system, but lends itself to overfitting if handled poorly. Fortunately the question specifies a set of values for a particular value to train on, else one could fall in the trap of over-fitting the desired outcome.

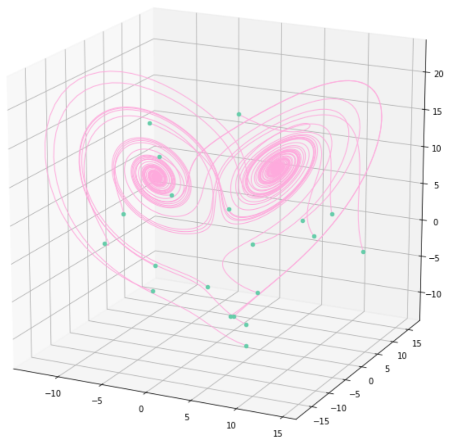

Figure 7: Random selection of Lorenz equation runs used to train the neural network.

For this problem, as specified by the question, three different ρ values were used. Each ρ has 128 different initial condition samples that each run for 801 time steps. See Figure 7 for a collection of iterations used to train.

There is a lot of data. The hyperCurse of hyperparameters was found, and despised. Using varying network architectures and training rates, training set sizes, and epochs it was found that Lorenz is a fan of deep, wide, and slow times. Successive trial and errors found the best results required a much lower learning rate, slowing down the time to train - this was concluded after observing chaotic loss values with occasional tantalising promise.

To promote the non-linear system expected; `sigmoid`, `tanh`, and `selu` activation functions were used. It's expected that further epochs might improve the model, but given an abundance of caution not to overfit, and an emerging end-of-term pseudo-apocalypse nihilism, the model was only trained for 256 epochs. See an example of results in Figure 8.

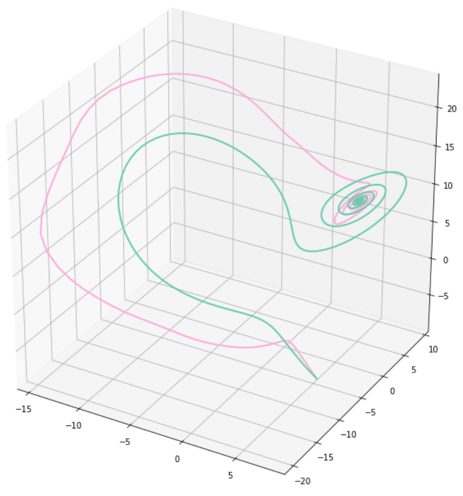

Figure 8: Random selection of Lorenz equation runs used to train the neural network.

Symptomatic dynamics are found, though there is very low accuracy. Without knowing the context the utility of the model is questionable. It is unlikely to fair well if tasked in a mechatronic rudder control, though may provide tolerable forecasting of orbit locations for social science qualitative models.

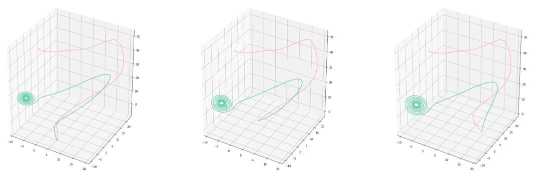

Figure 9: Iterating the model learnt with $\rho = 10, 28, 40$ with 3 random initial conditions for $\rho = 17$. The true solution is in mediumAquaMarine, and the predicted solution is in millenial pink.

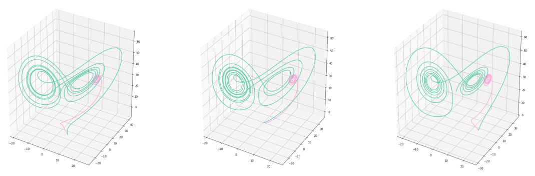

Figure 10: Iterating the model learnt with $\rho = 10, 28, 40$ with 3 random initial conditions for $\rho = 35$. The true solution is in mediumAquaMarine, and the predicted solution is in millenial pink.

Figures 9 and 10 show the model's predicted trajectories for different parameters. The goodness of fit seems, understandably, lower. Training on 3 parameters from an infinite continuum is not greatly generalisable in this problem. However it is interesting to see that the general initial direction is not completely random, but to some degree can approximate the true solution - and in particular for $\rho = 35$ we see tight orbits within the true attractors.

Sec V. Summary and Conclusions

Deep learning and neural networks are undeniably cool. Their applications to innumerable domains both in industry and academia as evidence. But in light of the previous homeworks, and course content it can't help but feel slimy.

Previously we have used *expert* domain knowledge to predict system structure, and fit historical data to find the best parameters. Deep learning instead phishes for many varaibles to populate a *dumb* architecture.

This approach requires resignation to compute - though to great affect. Relinquishing authority of expertise to the machine lets us infer complex systems beyond our comprehension in exchange for tuning of parameters.

The research areas involving dynamic construction, sparsity, and adversarial training all provide interesting approaches to the complaints I'm encountering. Critcism is founded in the difficulty of optimising training, but deep learning has only reached academic popularity recently. The modern deep learning resurgence started in 2006, it has been around 14 years. The Sugababes never knew a world with Paul Dirac, and yet Quantum Mechanics still suffers from his notation - by historical standards Deep Learning is approachable but is still in infancy.