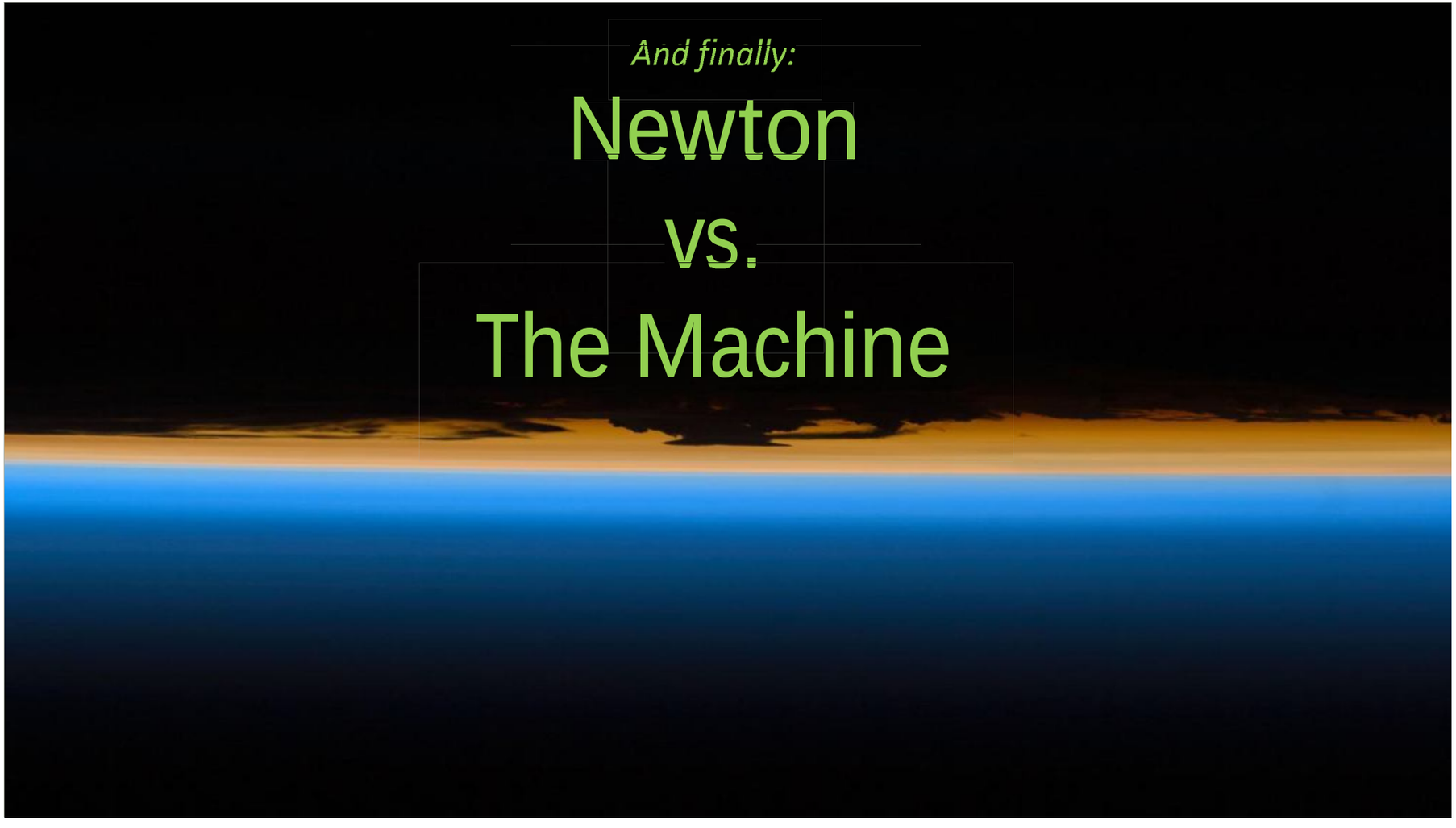

And finally:
Newton
vs.
The Machine

MNRAS **000**, 1–6 (2019) Preprint 17 October 2019 Compiled using MNRAS LATEX style file v3.0

Newton vs the machine: solving the chaotic three-body problem using deep neural networks

Philip G. Breen[1]★†, Christopher N. Foley[2] ★‡, Tjarda Boekholt[3] and Simon Portegies Zwart[4]

[1]*School of Mathematics and Maxwell Institute for Mathematical Sciences, University of Edinburgh, Kings Buildings, Edinburgh, EH9 3JZ*
[2]*MRC Biostatistics Unit, University of Cambridge, Cambridge, CB2 0SR, UK.*
[3]*Instituto de Telecomunicações, Campus Universitário de Santiago, 3810-193, Aveiro, Portugal*
[4]*Leiden Observatory, Leiden University, PO Box 9513, 2300 RA, Leiden, The Netherlands.*

Accepted XXX. Received YYY; in original form ZZZ

ABSTRACT

Since its formulation by Sir Isaac Newton, the problem of solving the equations of motion for three bodies under their own gravitational force has remained practically unsolved. Currently, the solution for a given initialization can only be found by performing laborious iterative calculations that have unpredictable and potentially infinite computational cost, due to the system's chaotic nature. We show that an ensemble of solutions obtained using an arbitrarily precise numerical integrator can be used to train a deep artificial neural network (ANN) that, over a bounded time interval, provides accurate solutions at fixed computational cost and up to 100 million times faster than a state-of-the-art solver. Our results provide evidence that, for computationally challenging regions of phase-space, a trained ANN can replace existing numerical solvers, enabling fast and scalable simulations of many-body systems to shed light on outstanding phenomena such as the formation of black-hole binary systems or the origin of the core collapse in dense star clusters.

Key words: stars: kinematics and dynamics, methods: numerical, statistical

1 INTRODUCTION

Newton's equations of motion describe the evolution of many bodies in space under the influence of their own gravitational force (Newton 1687). The equations have a central role in many classical problems in Physics. For example, the equations explain the dynamical evolution of globular star clusters and galactic nuclei, which are thought to be the production sites of tight black-hole binaries that ultimately merge to produce gravitational waves (Portegies Zwart & McMillan 2000). The fate of these systems depends crucially on the three-body interactions between black-hole binaries and single black-holes (e.g. see Breen & Heggie 2013A,B; Samsing & D'Orazio 2018), often referred to as close encounters. These events typically occur over a fixed time interval and, owing to the tight interactions between the three nearby bodies, the background influence of the other bodies can be ignored, i.e. the trajectories of three bodies can be generally computed in isolation (Portegies Zwart & McMillan 2018). The focus of the present study is therefore the timely computation of accurate solutions to the three-body problem.

Despite its age and interest from numerous distinguished scientists (de Lagrange 1772; Heggie 1975; Hut & Bahcall 1983; Montgomery 1998; Stone & Leigh 2019), the problem of solving the equations of motion for three-bodies remains impenetrable due to the system's chaotic nature (Valtonen et al 2016) which typically renders identification of solutions feasible only through laborious numerical integration. Analytic solutions exist for several special cases (de Lagrange 1772) and a solution to the problem for all time has been proposed (Valtonen et al 2016), but this is based on an infinite series expansion and has limited use in practice. Computation of a numerical solution, however, can require holding an exponentially growing number of decimal places in memory and using a time-step that approaches zero (Boekholt et al 2019). Integrators which do not allow for this often fail spectacularly, meaning that a single numerical solution is unreliable whereas the average of an ensemble of numerical solutions appear valid in a statistical sense, a concept referred to as nagh Hoch (Portegies Zwart & Boekholt 2018). To overcome these issues, the Brutus integrator was developed (Boekholt & Portegies Zwart 2015),

★ Authors contributed equally
† Contact e-mail: phil.breen@ed.ac.uk
‡ Contact e-mail: christopher.foley@mrc-bsu.cam.ac.uk

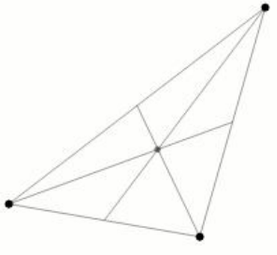
The Three-body Problem

You may have studied the two-body problem. One of its characteristics is that it is wonderfully analyzable for a variety of potentials.

The next step up the ladder of n-body problems is the three-body model. This has been an active area of research since Newton as it not only applies to such important cases as the Sun-Earth-Moon, but it also defies simple analysis.

We know that the general solution can not be represented by any closed solution (although there has been relatively recent work regarding infinite series).

Many of the greatest mathematicians have devoted efforts to the problem and some particular solutions were found by Newton, Jacob Bernoulli, Daniel Bernoulli, Leonhard Euler, Joseph-Louis Lagrange, Pierre-Simon Laplace, and others. Henri Poincaré's efforts were a large part of the foundation of modern chaos theory.



Wikipedia Commons

Numerically Challenging

Generating the training data here exemplifies how difficult it is to do this accurately. The "classical" method that they used (actually quite sophisticated, and a very modern code) uses an advanced numerical method and includes variable precision.

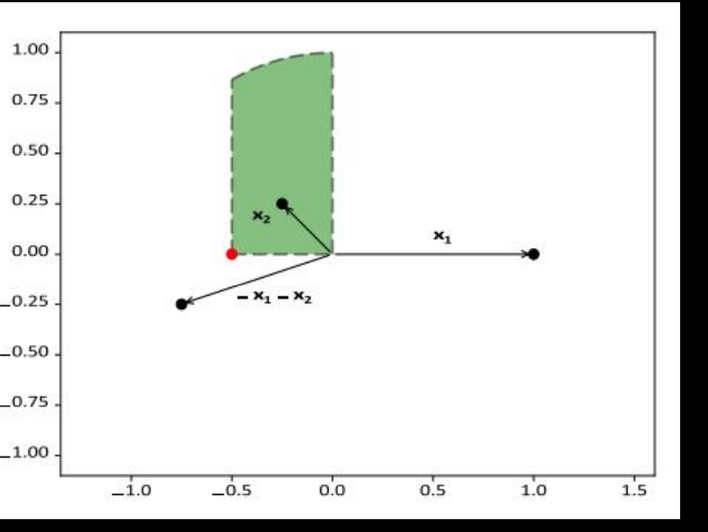
Despite its age and interest from numerous distinguished scientists (de Lagrange 1772; Heggie 1975; Hut & Bahcall 1983; Montgomery 1998; Stone & Leigh 2019), the problem of solving the equations of motion for three-bodies remains impenetrable due to the system's chaotic nature (Valtonen et al 2016) which typically renders identification of solutions feasible only through laborious numerical integration. Analytic solutions exist for several special cases (de Lagrange 1772) and a solution to the problem for all time has been proposed (Valtonen et al 2016), but this is based on an infinite series expansion and has limited use in practice. Computation of a numerical solution, however, can require holding an exponentially growing number of decimal places in memory and using a time-step that approaches zero (Boekholt et al 2019). Integrators which do not allow for this often fail spectacularly, meaning that a single numerical solution is unreliable whereas the average of an ensemble of numerical solutions appear valid in a statistical sense, a concept referred to as nagh Hoch (Portegies Zwart & Boekholt 2018). To overcome these issues, the Brutus integrator was developed (Boekholt & Portegies Zwart 2015), allowing for close-to-zero time-steps and arbitrary precision. Brutus is capable of computing converged solutions to any gravitational N-body problem, however the process is laborious and can be extremely prohibitive in terms of computer time. In general, there does not exist a theoretical framework capable of determining a priori the precision required to deduce that a numerical solution has converged for an arbitrary initialization (Stone & Leigh 2019). This makes the expense of acquiring a converged solution through brute-force integration unpredictable and regularly impractical.

A converged solution was acquired by iteratively reducing two parameters during integration: (i) the tolerance parameter (ϵ), controlling accuracy, that accepts convergence of the Bulirsch-Stoer multi-step integration scheme (Bulirsch & Stoer 1964) and; (ii) the word length (L_w) measured in bits, which controls numerical precision (Boekholt & Portegies Zwart 2015). Our ensemble of initial realizations all converged for values of $\epsilon = 10^{-11}$ and $L_w = 128$ (see Appendix A). Generating these data required over 10 days of computer time. Some initialisations gave rise to very close

Reducing the Parameters

For this restricted version of the problem (3 equal masses with zero initial velocities) we can use a little physics and reduce the number of parameters required to represent the initial configurations.

- Reference frame is center of mass
- X_1 is always at (1,0)
- Specify X_2's x,y coordinates
- X_3 is given by symmetry ($-X_1$-X_2)



We only need two parameters (X_2 x,y coordinates) to specify any desired initial condition!

If you recall our "sophisticated" view of neural nets as mappings from input vectors to output vectors, you can imagine we have simplified the task for our network a good deal.

Training Data

Our training data is going to have an input file with each sample consisting of an initial configuration and a timestamp for the desired final configuration.

The corresponding output is going to be the numerically generated final configuration in a reduced form: X_1 and X_2 positions (X_3 follows from symmetry).

train_X.csv

```
0.652344,-0.1648881236291216,0.9141665763358605
0.519531,-0.1648881236291216,0.9141665763358605
0.21875,-0.1648881236291216,0.9141665763358605
...
```

train_Y.csv

```
0.8724009514007633,0.03988576497011263,-0.1041928066877228,0.7964733750758993
0.9205503757602744,0.02465380676629187,-0.127317806825031,0.8407822474813996
0.9862529823431486,0.004226939675911646,-0.1584345882281039,0.9014472881328324
...
```

Our dataset will only contain the <3.9s class of data - or about 39% of the longest, 10s, trajectories mentioned in the paper.

Your Model

The paper describes the network used to generate the results. In particular, all of the hyperparameters.

This is everything you need to know to reproduce their results.

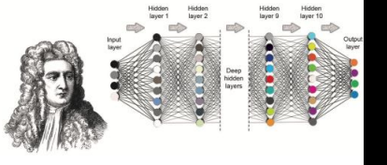


Figure 2. Newton and the machine. Image of sir Isaac Newton alongside a schematic of a 10-layer deep neural network. In each layer (apart from the input layer), a node takes the weighted input from the previous layer's nodes (plus a bias) and then applies an activation function before passing data to the next node. The weights (and bias) are free parameters which are updated during training.

counters, and computation of converged solutions in these situations is costly[1] (Boekholt et al 2019).

We used a feed-forward ANN consisting of 10 hidden layers of 128 interconnected nodes (Fig. 2 and Appendix B). Training was performed using the adaptive moment estimation optimization algorithm ADAM (20) with 10000 passes over the data, in which each epoch was separated into batches of 5000, and setting the rectified linear unit (ReLU) activation function to $\max(0, x)$ (Glorot, Bordes & Bengio, 2011). By entering a time t and the initial location of particle x_2 into the input layer, the ANN returns the locations of the particles x_1 and x_2 at time t, thereby approximating the latent analytical solution to the general three-body problem.

To assess performance of the trained ANN across a range of time intervals, we partitioned the training and validation datasets into three segments: $t \lesssim 3.9$, $t \lesssim 7.8$ and $t \lesssim 10$ (which includes all data). For each scenario, we assessed the loss-function (taken as the mean absolute error MAE) against epoch. Examples are given in Fig. 3. In all scenarios the loss in the validation set closely follows the loss in the training set. We also assessed sensitivity to the choice of activation function, however no appreciable improvement was obtained when using either the exponential rectified (Clevert, Unterthiner & Hochreiter 2011) or leaky rectified (Maas, Hannun & Ng 2013) linear unit functions. In addition, we assessed the performance of other optimization schemes for training the ANN, namely an adaptive gradient algorithm (Duchi, Hazan & Singer 2011) and a stochastic gradient descent method using Nesterov momentum, but these regularly failed to match the performance of the ADAM optimizer.

Results

You should be able to reproduce these same results for the <3.9s data.

You should include an error graph that looks very similar to theirs.

Reproducing some sample trajectories is optional, but recommended to make sure you really understand what you are doing. *This is worth a lot of partial credit if you need it!*

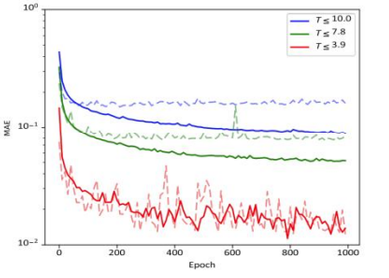


Figure 3. Mean Absolute Error (MAE) vs epoch. The ANN has the same training structure in each time interval. Solids lines are the loss on the training set and dashed are the loss on the validation set. $T \leq 3.9$ corresponds to 1000 labels per simulation, similarly $T \leq 7.8$ to 2000 labels and $T \leq 10.0$ to 2561 labels/time-points (the entire dataset).The results illustrate a typical occurrence in ANN training, there is an initial phase of rapid learning, e.g. âL̈ś100 epochs, followed by a stage of much slower learning in which relative prediction gains are smaller with each epoch.

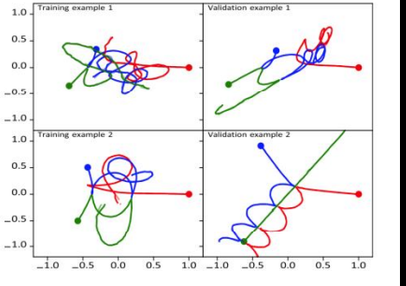


Figure 4. Validation of the trained ANN. Presented are two examples from the training set (left) and two from the validation set (right). All examples were randomly chosen from their datasets. The bullets indicate the initial conditions. The curves represent the orbits of the three bodies (red, blue and green, the latter obtained from symmetry). The solution from the trained network (solid curves) is hardly distinguishable from the converged solutions (dashes, acquired using Brutus (Boekholt & Portegies Zwart 2015)). The two scenarios presented to the right were not included in the training dataset.

Chaotic Motion

Chaotic motions like these are not only interesting and challenging, but they represent a great deal of everyday mechanics, not just double-pendulums. Most fluid dynamics (meteorology, etc.) is firmly in this category.

10 years ago we might have spent considerable time on this topic. Alas, it has been displaced by all of our very modern techniques. The good news is that these very modern techniques have given us exciting new tools for these classic problems.

Some flavor of the central characteristic of chaotic motion - a sensitivity to initial conditions - is explored here. Some very basic insight into how the trajectories diverge with consistency is seen here.

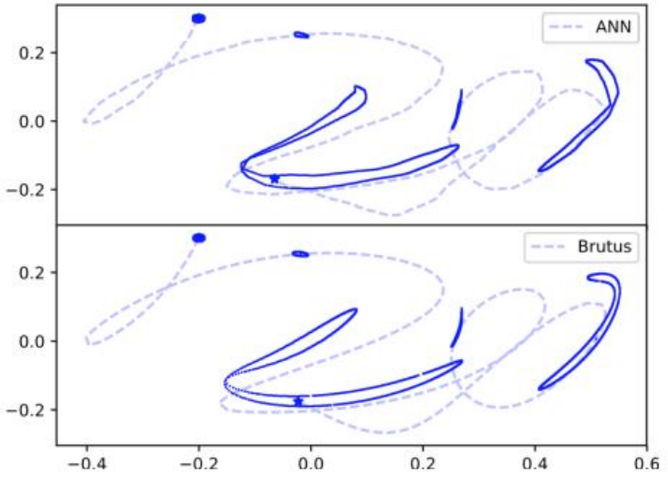


Figure 5. Visualization of the sensitive dependence on initial position. Presented are trajectories from 1000 random initializations in which particle x_2 is initially situated on the circumference of a ring of radius 0.01 centred at (-0.2, 0.3). For clarity, these locations were benchmarked against the trajectory of x_2 initially located at the centre of the ring (hatched line), the star denotes the end of this trajectory after 3.8 time-units. None of these trajectories were part of the training or validation datasets. The locations of the particles at each of five timepoints, $t \in \{0.0, 0.95, 1.9, 2.95, 3.8\}$, are computed using either the trained ANN (top) or Brutus (bottom) and these solutions are denoted by the bold line. The results from both methods closely match one another and illustrate a complex temporal relationship which underpins the growth in deviations between particle trajectories, owing to a change in the initial position of x_2 on the ring.

Conserved Quantities

Even complex systems may have conserved quantities. Ours certainly does, and it makes a great check of correctness.

Our network doesn't actually output velocities directly. The most obvious thing to do would be to derive these using the trajectories.

The paper discusses a network that actually generates velocities. It also discusses an improved algorithm that incorporates this into an improved error term.

There remain interesting areas of improvement to explore. That you could do!

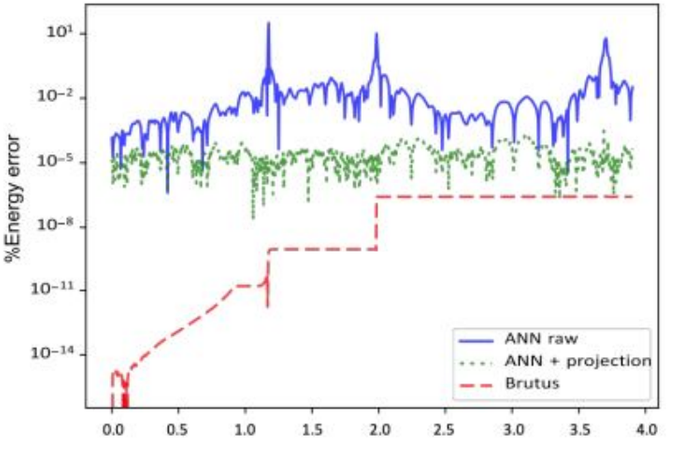


Figure 6. Relative energy error. An example of the relative energy error for a typical simulation. The raw output of the two ANN's typically have errors of around 10^{-2}, after projecting onto a near by energy surface the error reduces down to order 10^{-5}.

Assignment #6: Newton vs. The Machine

We are only concerned with the most accurate trajectories, the case of t < 3.9s. Otherwise the dataset is as described in the paper*.

The files are found at `~urbanic/LargeScaleComputing/Newton_vs_Machine`, along with a copy of the paper.

In this format, the data can be easily read. One way to get it into a TensorFlow compatible numpy array is with

```
import pandas as pd
inputf = pd.read_csv("train_X.csv").values
target = pd.read_csv("train_Y.csv").values
```

but feel free to do something else.

I used the AI module for my own tests (as opposed to the singularity container I recommended for the MNIST work). Either should work, but you can use the AI module with a simple

```
module load AI
```

and then just start your python shell or run your python script and TensorFlow will be there.

* *We did eliminate two corrupted trajectories. You won't notice unless you count all the samples.*

Assignment #6: Newton vs. The Machine

To reproduce the results shown in Figure 3 of the paper (for t < 3.9s) you will need to run 1000 epochs. We found this to take about 100 minutes.

The good news is that you shouldn't be doing much of a hyperparameter search as the paper is handing you all of the effective values. If you nail it, you will only need one run.

I don't expect you to nail it the first time. So, pay attention and if the error isn't looking something like Figure 3, kill your run and try again.

The paper has the (very small) validation split of only 1% of the data. You can split this out of the given data yourself (use Spark!). Or, this might be a good time to learn about the `validation_split` option in `model.fit()`.

Assignment #6: Logistics

Deadline: December 5th, last day of class. Before we start the lecture.

Points: 10 0points.

Format: Email me your TensorFlow script as an attachment and the detailed results in the body of the email. This includes at least the final test accuracy and the error graph. You may also want to include comments as your incremental efforts along the way will count for credit.