CS230

CS 230 Project Final Report: Inverting Yarn-Level Cloth Relaxation

Jonathan Leaf
Stanford University
jcleaf@stanford.edu

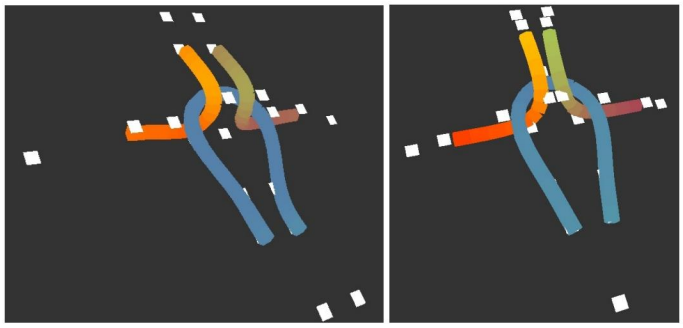
Abstract

To calibrate physics-based yarn-level cloth relaxation, I built a deep neural network to predict physical simulation parameters from a relaxed yarn configuration. Each input is a yarn configuration: a set of ordered spline control points that defines yarn curves. Each output is a physical parameter that affects the bending energy of the yarns. The network predicts the correct physical parameter within ± 0.1 with 93.6% accuracy on the test set.

1 Introduction

Yarn-level cloth relaxation is the process of taking a synthetically generated yarn topology and running physics-based simulations to generate a feasible or physically plausible configuration. Primarily used in computer graphics, this technique maps an initial configuration and a set of physical parameters to a physically plausible yarn configuration. Figure 1 shows a simple knit-loop structure before and after relaxation.

Figure 1: Comparison of yarn state before and after relaxation.



(a) Before Relaxation (b) After Relaxation

When attempting to calibrate relaxation models, one must use measurements of actual knitted or woven fabrics. However, measured yarn data can be viewed as in a "relaxed" state, which hides much of the critical physical parameters that were required to get the yarn into its relaxed state.
This calibration process is difficult. The many parameters of relaxation, and the many permutations of possible initial conditions make learning this space immensely difficult. Thankfully, because clothing

CS230: Deep Learning, Winter 2018, Stanford University, CA.

has repeatable patterns and shapes, we can learn on very small fixed-size problems and generalize this information to other parts of the model.
In this work, I train a neural network to learn the inverse of yarn-level cloth relaxation on a very small problem, going from output yarn data to a physical parameter that affects the yarns bending energy. Using the relaxation engine as a black box, I invert it's input and output data to learn the inverse mapping, which will help when I calibrate my relaxer against measured data. This work will help to calibrate relaxation methods to real clothing, enabling more realistic textures for 3D graphics in films and movies.

2 Related work

The simulator used in this work is directly taken from [KJM08], which defined yarn-level cloth relaxation. The work [KJM10] improved efficiency of the simulator and tweaked the yarn model.

This work is unique due to the novelty of applying deep learning to yarn-level cloth relaxation. That said, the idea to learn parameters from measured data is not new. There are works that exist to infer physical parameters or interactions from video data. [MTM] successfully analyze video data to reconstruct collisions between different objects. Using deep learning to collect physical parameters from measured data is an important step in connecting real objects to our physical models and virtual reality.

3 Dataset

Without access to measured data for this work, I opt to generate data using a simulator to train and test the network. I am using a research simulator to generate cloth simulation data to feed into the network. Under this, my dataset size is constrained by compute time and the time to set up more examples.

Because traditional neural networks are trained on fixed-sized inputs, I pick a single topological pattern with a fixed number of degrees of freedom and permute it's bending parameter to generate a sufficiently large data set for training. In general, permutations could include translation, rotation, random position permutations that do not change yarn topology, scaling multiple physical constants such as bending and stiffness parameters, scaling rest-length parameters, and scaling control point masses. To keep the scale of this work feasible, I only sweep the bending parameter to generate examples.

The dataset is separated into inputs and outputs:

1. The input dataset consists of real-numbered values for each x,y,z position of each control point in the "relaxed" state.
2. The output dataset consists of the physical bending parameter used to run the model. We sweep this bending parameter to collect a large enough sample size.

The dataset is made up of 10,000 examples. The inputs are made up of the relaxed state of the model, with 63 real number values corresponding to the xyz of each of the 21 points of the 3 yarn system as seen in in Figure 1. The output is a single real-number value corresponding to a physical parameter that scales the bending energy of each yarn. To ensure a fast learning time, I normalized the inputs to have zero-mean and unit variance. I split the data into 90%/10% train/dev sets.

4 Method

I used a traditional fully-connected neural network. The network used 2 hidden layers and a linear/no-activation output layer. Both hidden layers used the RELU activation function. Both hidden layers had the same number of nodes as the dimensions of the network input. Mathematically, each hidden layer has the form:

$$A^{[l]} = \sigma_{RELU}(W^{[l]}A^{[l-1]} + b^{[l]})$$

Where W is the weight parameter matrix, A is the activation of the previous layer, b is the bias parameter matrix, σ_{RELU} is the activation function. As my loss function, I used a sum of squared

difference loss function (which is the squared L2 difference norm). Mathematically:

$$L = \sum (y^{(i)} - \hat{y}^{(i)})^2$$

The neural network has the following architecture:

1. 63 inputs corresponding to 21 points.
2. 63 node fully-connected hidden layer using RELU
3. 63 node fully-connected hidden layer using RELU
4. 1 node output layer. No activation on the final layer.

5 Experiments/Results/Discussion

I chose to use an Adam Optimizer with the following parameters:

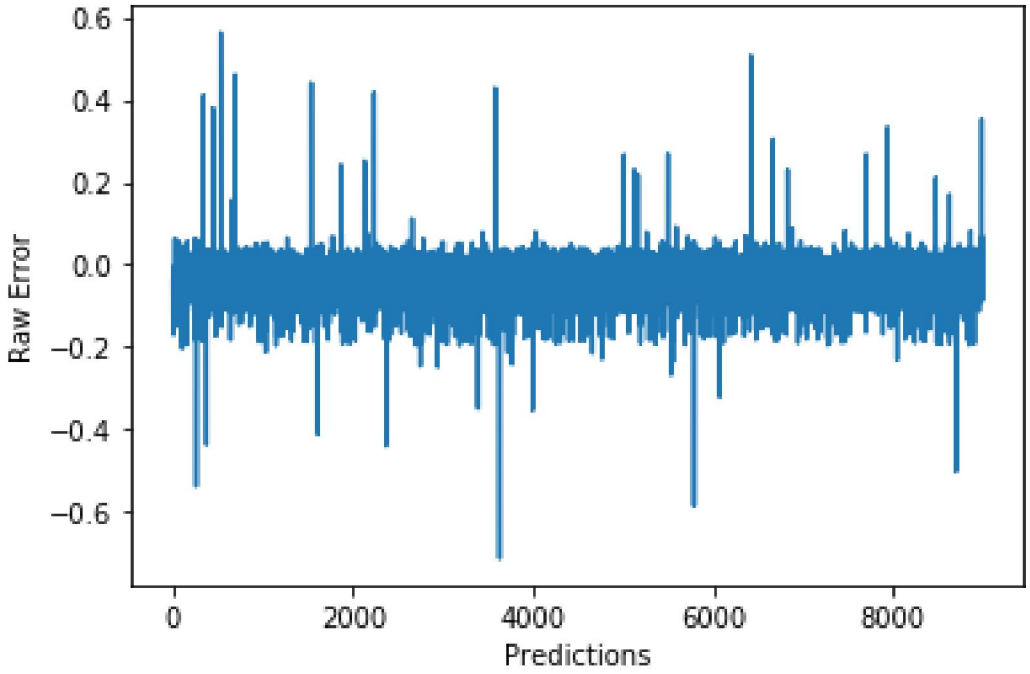
1. Learning Rate: $\alpha = 0.005$.
2. $\beta_1 = 0.9$
3. $\beta_2 = 0.999$

I used a mini-batch size of 300 examples. My primary metric for evaluating the performance is the accuracy of the predictions (M_{acc}), which is defined as the following:

$$M_{acc} = \frac{\sum_{i=0}^{m} 1\{|y^{(i)} - \hat{y}^{(i)}| < 0.1\}}{m}$$

'm' is the total number of training examples in the evaluated set of examples. is the Depending on how you select the counting threshold you will get different percentages with this accuracy function. The raw training error for all training examples can be seen in Figure 2

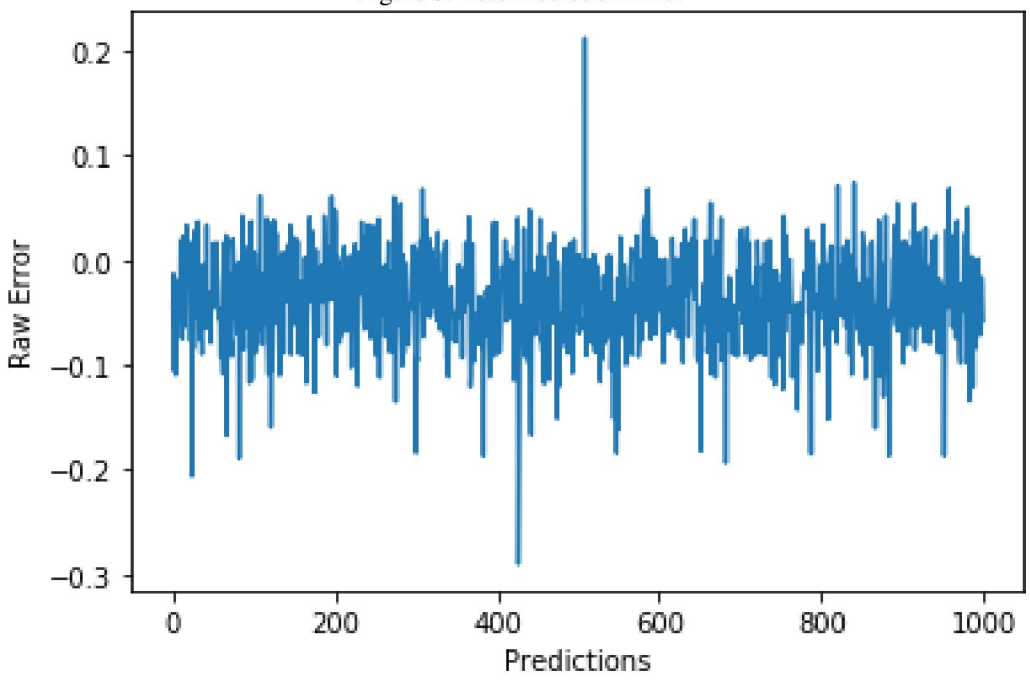
Figure 2: Training Prediction Error



I selected the threshold to be 0.1 and report an accuracy of 91.75% on the training set. To illustrate this decision, it is valuable to look at the extremes. If we set the threshold to 1.0, we would report

an accuracy of 100%, which may be misleading which is also uncharacteristic of the overall performance. On the flip side, if we used a threshold of 0.01, the accuracy becomes 11.6%. The physical parameter being estimated in the dataset is sampled at intervals of 0.01, so we would expect the neural network to not be able to reduce its estimate uncertainty close to the sampling frequency. If the dataset provided samples at 0.001 intervals, we should expect the accuracy of the network to improve.
Figure 3 contains the raw test error for all test examples. The accuracy metric is also applied to the test error, and reports an accuracy of 93.6% with a threshold of 0.1.

Figure 3: Test Prediction Error



The network does marginally better on the test set than the training set overall. I suspect this is because the network doesn't have enough layers to capture the "harder" examples in the training set, and the test set happens to have more "easier" examples.

6 Conclusion/Future Work

Having established this initial network, I have provided a proof of concept for learning physical parameters from yarn models. For a fixed-sized pattern, and knowledge of the initial condition, it is possible to learn the mapping from relaxed/simulated yarn data to a physical bending energy parameter. With more computational time, I could generate more data and use a deeper network to learn coupled physical parameters from relaxed yarn data. Also, moving from a small model to a larger model would help argue that this can be achieved on different knitted/woven patterns. Additionally, estimating the initial state instead of just physical parameters is computationally taxing, but will enable a more generalizable learning system.

Contributions

I am the only contributor to this work.
Link to Github Repo: https://github.com/jcleaf/cs230
I used Pandas, Tensorflow, Numpy, and scikit-learn to help with data management. I downloaded a fully-connected neural network implementation from https://github.com/lucko515/fully-connected-nn using an MIT license to get started, and used some of the data processing code from this project to parse my data.

References

[KJM08] Jonathan M. Kaldor, Doug L. James, and Steve Marschner. Simulating knitted cloth at the yarn level. *ACM Transactions on Graphics*, 2008.

[KJM10] Jonathan M Kaldor, Doug L James, and Steve Marschner. Efficient Yarn-based Cloth with Adaptive Contact Linearization. 2010.

[MTM] Aron Monszpart, Nils Thuerey, and Niloy J Mitra. SMASH: Physics-guided Reconstruction of Collisions from Videos.