STYLE TRANSFER WITH CNN

Casado Herraez, Daniel
shreyjoshi2004@gmail.com

Gatri, Wathek
shreyjoshi2004@gmail.com

Gimenez Bolinches, Andreu
shreyjoshi2004@gmail.com

March 25th, 2019

ABSTRACT

Abstract

***K*eywords** Artificial Neural Network · Curve Fitting · Dataset · Polynomial · Regression

1 Introduction

2 Background Research

2.1 A Brief Introduction To The Artificial Neural Network

Artificial Neural Networks (ANNs) are computational systems loosely modelled after the networks of neurons in the animal brain. Their versatility and ability to solve problems not easily able to be expressed algorithmically comes from the **fundamentally** different way they approach problems. Examples of these problems include, but are not limited to

- Image Recognition and Classification
- Function Modelling and Curve Fitting
- Image Restoration
- Language Translation
- Voice Synthesis and Recognition

A notable similarity among these five problems is that they are all non-trivial pattern recognition problems. In this project we investigate Artificial Neural Networks as they relate to the preantepenultimate point on this list, Function Modelling and Curve Fitting.

2.2 Artificial Neurons

An Artificial Neural Network is simply a network of **artificial neurons**, divided into various **layers**. There are two major types of neurons: **perceptrons** and **sigmoid neurons**.

2.2.1 Perceptrons

Perceptrons are the simplest types of neurons, they receive binary inputs and produce a binary output. To compute this output we introduce **weights**, real numbers signifying the strength of the effect of a particular input on the perceptron's output.

The output, or **activation**, of a perceptron is determined by whether the value of the **weighted sum** $\sum_{i=1}^{n} x_i w_i$ is greater than some **threshold** value. More explicitly,

$$\text{Output} = \begin{cases} 0 & \text{if} \quad \sum_{i=1}^{n} x_i w_i \leq \text{threshold} \\ 1 & \text{if} \quad \sum_{i=1}^{n} x_i w_i > \text{threshold} \end{cases} \tag{1}$$

To make the notation less cumbersome, let x and w be the vectors of inputs and their corresponding weights, respectively and define a **bias** value $b = -\text{threshold}$. Then we get

$$\text{Output} = \begin{cases} 0 & \text{if} \quad w \cdot x + b \leq 0 \\ 1 & \text{if} \quad w \cdot x + b > 0 \end{cases} \tag{2}$$

2.2.2 Sigmoid Neurons

For our network to be able to learn in a predictable manner, we need the output(s) of the networks to be continuous functions of the input(s), however this is not the case with the perceptron model. Hence, we introduce a new type of neuron, the sigmoid neuron.

Sigmoid neurons function exactly the same as perceptrons, except the activation of the sigmoid neuron is not a piecewise function, it's rather defined by $\sigma(w \cdot x + b)$, where w and x are the vectors of the inputs and their corresponding outputs respectively, b is the bias, and $\sigma(z)$ (the sigmoid function) is defined as

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{3}$$

2.3 Artificial Neural Network Architecture

Artificial Neural Networks are divided in layers, with an arbitrary number of neurons in each layer. The first and last layers are the input and output layer respectively, and all layers in between are called **hidden layers**.

It is evident that the outputs from layer l are the inputs to layer $l + 1$. This repeated process of taking the outputs from one layer and inputting them to the next is called **forward-propagation**. Neural networks requiring only a forward-propagation system to find the output are called **feed-forward** neural networks.

When numbers are inputted to the network, they're multiplied by their corresponding weights and inputted to the neurons of the second layer. The weighted sum of each of the neurons is then added to their bias and put into the sigmoid activation function. These values are now multiplied by the weights of the connections between the second and third layers and the process is repeated until the last layer.

If we let

- a_j^l denote the activation of the jth neuron in layer l
- $w_{j,k}^l$ denote the value of the weight connecting the jth neuron in layer l and the kth neuron in layer $l - 1$
- b_j^l denote the bias of the jth neuron in layer l
- n_l denote the number of neurons in layer l

We can define a universal equation to find the activation of any neuron in our network

$$a_j^l = \sigma\left(\left[\sum_{k=1}^{n_{l-1}} w_{j,k}^l a_k^{l-1}\right] + b_j^l\right) \tag{4}$$

2.4 Training An Artificial Neural Network

The training process of an ANN is highly variable when it comes to the specifics, but the general structure is the following

1. Define a cost function (also error, loss, or objective function) of the network's parameters
2. Initialize the network with random parameters
3. Iterate a series of steps which changes the network parameters in such a way as to decrease the cost function

2.4.1 The Mean-Squared-Error Cost Function

To be able to train our network we must define a **cost function** (or **error function**), showing how well our network is performing. This function takes in the parameters (weights and biases) of the network and a set of **training data**, which is data from which our network must 'learn', or to be less ambiguous, it is the data from which our network must generalize. In short, it is, quite literally, the training data. One such cost function is the **Mean-Squared-Error (MSE)** cost function, or quadratic cost function. The MSE cost function is defined as

$$C(w, b) :\propto \sum_x ||y(x) - a^L(x)||^2 \tag{5}$$

where we write the desired output and actual output of the network as $y(x)$ and $a^L(x)$, respectively, as they are functions of the inputs, x. Usually the constant of proportionality is $1/2n$ but it does not matter as much as the general form of the equation. The MSE cost function is a good way of determining the error of a network as it gives a high value when even a single desired output is not close to the actual output since the raw difference is squared, and when the two outputs are close together, the raw difference (which is small) is squared to become even smaller, so that point is then basically ignored.

2.5 Engineering Goal

The main purpose of this project is to demonstrate the effectiveness of Artificial Neural Networks as a curve fitting model, and in particular, display its superiority over traditional methods such as polynomial regression.

There are three major criteria for this ANN model to satisfy:

1. Very High Scalability
2. Good Flexibility Between Accuracy and Speed
3. Insensitivity to Noise (An Avoidance of Overfitting)

Immediately from our preliminary research we can see that ANNs possess inherent properties which resolve much of the design criteria.

1. The issue of scalability is very easily avoided just by considering the structure of neural nets. If we want to curve fit a function $f : \mathbb{R}^n \to \mathbb{R}^k$ then we simply construct a network with n neurons in the input layer and k neurons in the output layer and train it with training data.
2. Since ANNs train by an iterative process, there is a clear speed-accuracy trade-off, and thus training can be stopped at an arbitrary point in time to sacrifice accuracy for speed or vice versa. This ensures that a researcher can perform a quick approximation of lengthy analysis of any dataset.
3. There exist regularization methods to lower the responsiveness of the training algorithm to random noise in the training data, hence decreasing the effects of overfitting.

2.6 Virtual Materials

There were no physical materials in this project, but a multitude of virtual materials were used, which include

- Python 3.7.0 programming software
- Numpy Library for fast linear algebra
- Matplotlib library for easy graphing
- Sci-kit learn library for neural network construction

2.7 Procedure

There were three programs developed during the duration of this project, which were for the purposes of

- Data Collection for the ANN model
- Data Collection for the Polynomial model
- Comparison Program which Graphs and a trained ANN and polynomial along with Training and Test data

1. Construct a three layer ANN of architecture $[1 - 2k - 1]$ with randomly initialized parameters/Select a polynomial order k.
2. Select a function, $f(x)$ from the above list from which to generate training data
3. Construct a new function $g(x) := f(x) + m$ where m denotes a random 10% error based on the normal distribution
4. Generate a training dataset of 15 sample points from $g(x)$
5. Train the ANN model/polynomial model with the training dataset
6. Once the model has been trained, obtain data on
   - Time Elapsed Training
   - Mean-Squared-Error
   - Average Percent Accuracy
   - Number of Epochs Elapsed (ANN only)

   where accuracies are measure with respect to the function $f(x)$.
7. Save the aforementioned data to an external text file
8. Perform 30 repetitions of steps 2-6 for each function
9. Perform one repetition of steps 1-7 for each $k \in \{1, 2, 3, ..., 10\}$
10. Summarize resulting data (take averages)

2.8 Major Documentation Of Progress

There emerged 4 major versions of the ANN program out of the continuous process of testing and bettering the design. Below is a comprehensive list of the version name and the changes associated with it.

- Version 1 - Basic ANN Mechanics: Training and numerical output of ANN based on input after training.
- Version 2 - Introduction of Polynomial Regression: Polynomials as a comparison benchmark and graphing of predictions after training
- Version 3 - Epoch Checking: An acceptable accuracy level is set and after each epoch of training, the ANN model's accuracy is compared to the acceptable accuracy. If it is greater, then training terminates.
- Version 4 - Update of Accuracy Algorithms: Percent accuracy is introduced and the MSE is updated to be faster
- Version 5 - Learning Rate Decay: This is the latest update and has not been tested yet. The main addition is the incorporation of a system that stores the parameters of the ANN of the previous epoch, so that if accuracy decreases between two epochs, the code reverts back to the older parametric settings, redefines the learning rate as 2/3 of itself, and continues training. This allows for the existence of learning rate decay without another hyperparameter.

A much more inclusive documentation can be found in the project written logbook.

3 Data

The cumulative data acquired can be summarized with the following graphs:

4 Conclusion

4.1 General Project Conclusions

In this project the plausibility of using Artificial Neural Networks (ANNs) as a regression model was tested. It was found that three-layer ANNs usually took longer than polynomial regression and had roughly similar accuracy, however, their rate of overfitting was incredibly lower and their usability was remarkably higher than polynomials. It was also observed that they were much less reliant on initial conditions, unlike polynomials which were very sensitive to their degree, and for this reason, ANNs are so much more resilient than their ANN counterparts. It was observed that in a three layer network, the optimum number of neurons for speed was around 12, and although there was an improvement of accuracy as the number of neurons increased, there was no big improvement after 14 neurons It is theorized that using deeper neural networks will greatly improve performance due to more layers of complexity, allowing the network to model more complicated datasets. This might introduce new problems, however, such as overfitting, but these can be solved with regularization, altering the activation function, and other techniques. It is intended to continue development of the ANN model and perform testing on multi-dimensional datasets to investigate its scalability. The full potential of ANNs as a curve fitting model was not revealed, but rather previewed in this project. It is conjectured that it will however be revealed upon further development and testing upon multivariate datasets with deeper networks, as this is where we can no longer apply standard methods with confidence.

4.2 Implications Of Data

A Two Sample T-test was conducted on the lists of the different Mean-Squared-Errors for the ANN and polynomial models for each individual class of functions. The average p value over these functional classes was computed to be 0.0370, meaning the data was statistically significant.

The data itself strongly implies a number of results:

The pair of activation function and loss minimization method that yielded the highest accuracy (lowest MSE) was hyperbolic tangent + LBFGS.

Testing different network architectures with hyperbolic tangent activation function + LBFGS yielded the result that 17 neurons tended to be optimal for curve fitting 2D datasets around the complexity of those used in this project.

From qualitative analysis of the graphs produced comparing the predictions of the models, it was found that ANNs were remarkably better for extrapolation than polynomials.

It was also seen from qualitative analysis that ANNs succumbed less to overfitting and hence were more usable, even when the number of neurons in the hidden layer was ridiculously large.

For these reasons, the researcher concludes that Artificial Neural Networks are superior to polynomials for curve fitting.

Unlike the polynomial graphs, there exists variation in the ANN graphs as parameters are initialized randomly (local minima)

4.3 Real-World Applications

4.4 Error Analysis

The following are some choices that may have affected the results:

- There were five functions used for testing the models. Using more functions to test the two models would have caused more accurate results since they would model a much wider variety of graph shapes.
- The choice of functions. Sufficiently high degree polynomials fit every function with a high degree of accuracy (> 90%) except for the asymptotic Function 2 due to the fact that polynomials have poor asymptotic properties. The representation of asymptotic function used while testing may not accurately reflect the distribution of data in the real world which models an asymptotic pattern, and for that reason the data could have been skewed. This problem can be solved in the future by increasing the diversity of the functions used.
- The constant choice of η, the learning rate. For different architectures, the optimum learning rates can be different and therefore the data might have been skewed towards the architectures for which $\eta = 6.5$ is the optimum learning rate. This issue can be solved in the future by incorporating a random hyperparameter search algorithm for optimizing the learning rate based on the architecture.