Using Neural Networks to Classify PDEs

Julia Balukonis[1], Sabrina Fuller[2], and Haley Rosso[3]

2019 Mathematics REU at The Georgia Institute of Technology

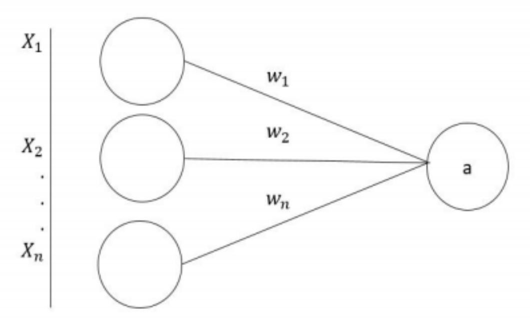
Abstract

We designed two neural networks that can learn how to classify three different types of partial differential equations (PDEs). Our data consists of numerical solutions to three categories of PDEs: Burger's, Diffusion, and Transport equations. Using TensorFlow and the Keras library, we performed two tasks - the first a binary classification of Burger's and Diffusion equation data, and the second a multi-label classification incorporating the Transport Equations as well. Our binary classification network requires vector labels to perform efficiently. Furthermore, our tertiary classification network continues to show that vector labeling provides the most accurate predictions. Our networks consistently make more accurate classifications and predictions than other classification tools, particularly Classification and Regression Trees (CART) and Support Vector Machine (SVM).

Introduction

A neural network is a computing architecture that models the relationship between the input data and its labels by composing linear and nonlinear activation functions. Appropriately designed, it can match data to its label as closely as possible.

The diagram below depicts a simple visualization of a neural network.



The *activation function* at the k^{th} neuron in the $l+1$ layer would be denoted as

$$a_k^{l+1} = \sigma(z_k^{l+1})$$

where

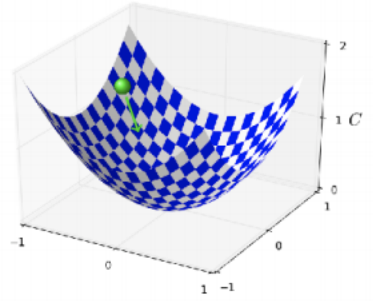
$$z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1}$$

The *cost function* is the difference between what the neural network outputs and the desired value. We notate this as

$$\mathcal{L} = \frac{(a-y)^2}{2}$$

Neural Network Architecture

Gradient descent can be imagined as a ball rolling down a valley; as it descends, it continually travels in the steepest direction, much like finding a local or global minimum in a function.



A neural network minimizes the loss by updating the weights and biases between its nodes, notated as:

$$b_j^l \leftarrow b_j^l - \eta \frac{\partial \mathcal{L}}{\partial b_j^l} \text{ and } w_{jk}^l \leftarrow w_{jk}^l - \eta \frac{\partial \mathcal{L}}{\partial w_{jk}^l}.$$

Backpropagation

Backpropagation is the process of a network first running untrained and then correcting itself based on its own miscalculations in order to minimize cost. This is done by finding the partial derivative of the previous layer's cost function with respect to the weights and biases. The adjustments in loss begin in the last layer L and proceed to the previous layer $L-1$ until it reaches the initial layer of the network.

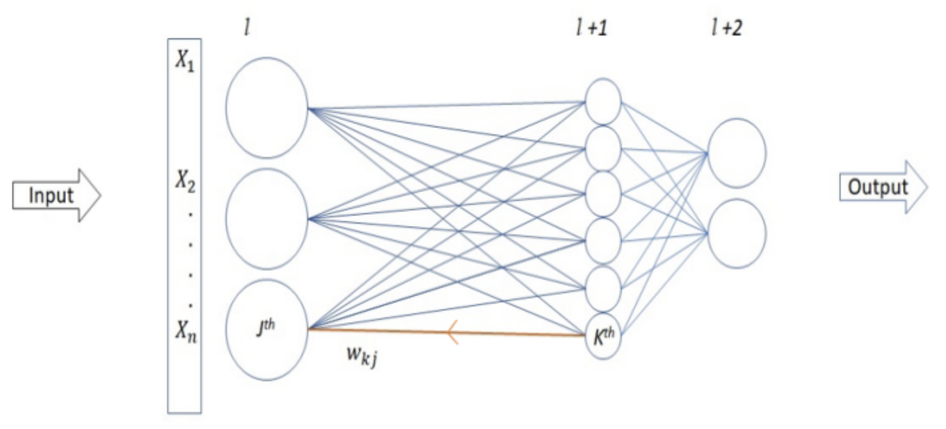
$$\frac{\partial \mathcal{L}}{\partial w_{jk}^L} = \frac{\partial \mathcal{L}}{\partial a_j^L} \cdot \frac{\partial a_j^L}{\partial z_j^L} \cdot \frac{\partial z_j^L}{\partial w_{jk}^L}$$

$$\frac{\partial \mathcal{L}}{\partial z_j^L} = \frac{\partial \mathcal{L}}{\partial a_j^L} \cdot \frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial \mathcal{L}}{\partial a_j^L}\sigma(z_j^L)$$

Now, for an arbitrary middle layer:

$$\frac{\partial \mathcal{L}}{\partial z_j^l} = \sum_k \frac{\partial \mathcal{L}}{\partial z_k^{l+1}} \cdot \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k w_{kj}^{l+1} \frac{\partial \mathcal{L}}{\partial z_k^{l+1}}\sigma(z_j^l)$$

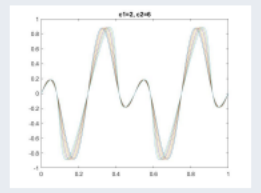
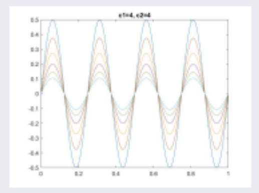
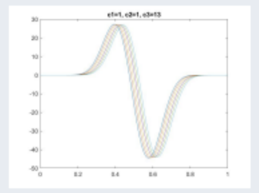
Backpropagation in a multi-layer neural network can be visualized in the figure below.



Applications

We devised two experiments to design this feed-forward network, the first a binary classifier of Burger's and Diffusion equations, and the second a network able to identify 3 partial differential equations. Both experiments were conducted using TensorFlow and its Keras Library.

PDE General Forms

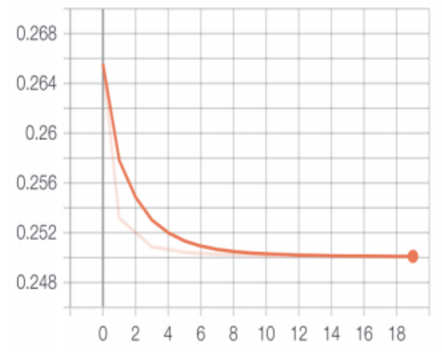
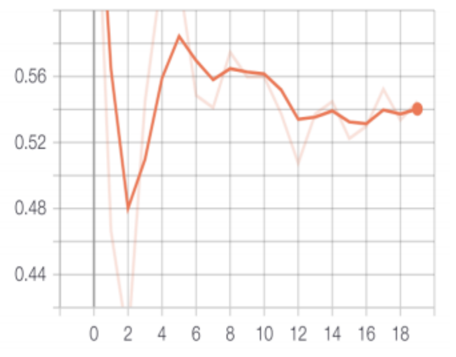
Burger's:	Diffusion:	Transport:
		
$u_t = -uu_x$	$u_t = 0.1u_{xx}$	$u_t = -u_x$

Our network consisted of 4 layers activated by the ReLU, Tanh, and Softmax functions. To observe the network's loss, we used the mean absolute error function. To then optimize the network, we implemented a learning rate (η) of 0.0015.

Layer	Number of Neurons	Activation Type
Input Layer	400	ReLU
2	150	Tanh
3	50	ReLU
Flatten	n/a	n/a
Output Layer	n	Softmax

Binary Classification

After 20 epochs while using labels of 0 and 1, this model minimized loss at approximately 25% (left figure below). However, the accuracy oscillated over the iterations (right figure below).

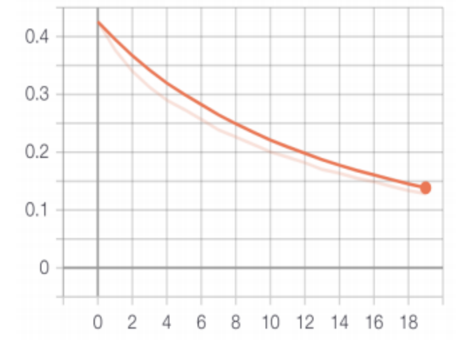
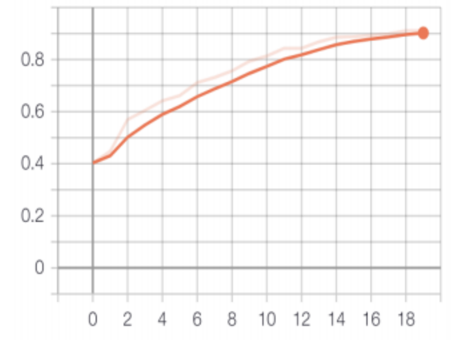



Multi-Label Classification

After a total of 20 epochs, with labels of

$$\begin{bmatrix}1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\end{bmatrix},$$

this model was able to minimize cost to approximately 13% (left figure below) and maximize accuracy at approximately 91% (right figure below) during this run.

Other Machine Learning Methods

We compared our neural network results with CART and SVM. Both methods returned accuracies of approximately $43-93\%$. A majority of these predictions were about equal the probability of flipping a coin.

Acknowledgements

We would like to thank our advisors, Dr. Micheal Lacey, Dr. Wenjing Liao, and Dr. Hao Liu for their immense help throughout the REU. We would also like to thank the National Science Foundation and The Georgia Institute of Technology for their funding and support.

References

[1] Michael Nielson. "Neural Networks and Deep Learning ". The World Wide Web. http://neuralnetworksanddeeplearning.com/

Contact Information: [1]Providence College, jbalukon@friars.providence.edu; [2]John Tyler Community College, sf8ez@virginia.edu; [3]University of Houston, hrosso@uh.edu