Spencer Holm

1st May 2019

AI-practicum

Neural Net to Predict Avalanches

The problem was the prediction of avalanches using a neural network to take in seemingly similar data and output whether or not an avalanche is possible. Meaning that given characteristics of snowpack would an avalanche present itself before it propagated. The methods used for the prediction would be a neural network.

A neural network is a machine learning algorithm where various input data is moved through nodes and weights to find the solution or expected result of the input. These nodes and edges form a weighted graph where data is meant to move forward and not backward to make a prediction. There are three main components to a neural net. First is the input layer which is as large as the number of different data points to be involved with determining a result. The second layer is the hidden layer which connects the input nodes to the output. The hidden layer is capable of being several layers of nodes and networks with multiple layers are referred to as deep learning networks. However, for many applications, multiple hidden layers would be too complicated for the problem being solved, and usually, only one hidden layer is implemented in a neural net. The final layer is the output layer where there can be one to multiple nodes each classifying a different possible solution [3].

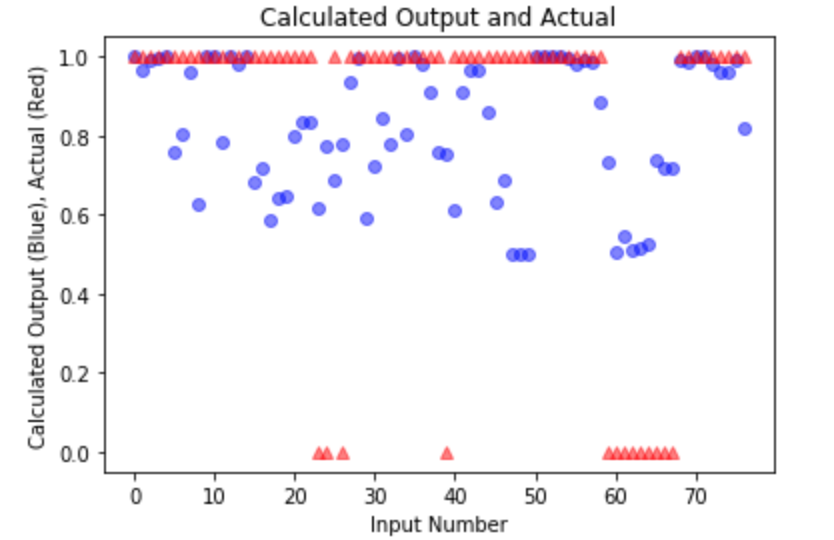
This would then require the edges and nodes weights to be set the first-time processing data correctly. Instead, the neural network iterates backward updating the weights. This is done by having a training set of data which has a known correct value and allowing for a comparison between the calculated value and the actual value. Then the weights are adjusted using the difference of the calculated value and the actual value until a minimum is reached and then forward propagation through the network will result in an accurate value [1].

Forward propagation of a neural network involves summing the weights multiplied by the input values for each node. Then passing the summed value through an activation function. The activation function is designed to introduce non-linearity to a neural network and works as a type of classifier of the value being presented at each node. This activation function can make it possible for a neural network to take number data and result in a classification result such as true or false in the form of a one or a zero. The many types of activation functions such as sigmoid, tanh, and step all have different uses, and they can be necessary for the type of problem a neural network is trying to solve.

The implementation along with this report has a simple structure of two input nodes and three hidden nodes with a single output node. This graph like structure was implemented with matrices along with NumPy to speed up computation through passing inputs as a matrix. The reason for limiting to two input nodes is the actual snow data being relatively inconsistent and when training to build a network with more input nodes the accuracy was decreased in the results after the training data was run through the network. The data and training data were the same making it more obvious when a larger number of input nodes were not working.

Backpropagation was done using a gradient descent technique to find a minimum to the updated weights that would result in better fitting results and have less possibility of increasing error in the network. This minimizing was done using SciPy and the optimization library which includes a function called Broyden-Fletcher-Goldfarb-Shanno which is an algorithm for solving iterative optimization problems [2]. The function is used when iterating over the training data and backpropagating to update the weights through this method until it finds an optimum set. Then the weights are permanently updated so as the forward propagate input through for an accurate result. The neural network implemented here does not also backpropagate after the training set and will keep the weights assigned during the training phase until the net is reconstructed.

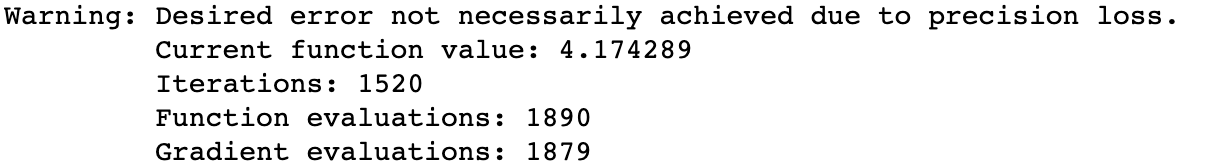
The results of the neural network showed that the snowpack measurements might not have been a good indicator of possible avalanches.



Plot : Neural net with over 1500 iterations

This plot shows the actual values of whether an avalanche was capable of being propagated with the calculated values from the neural network. This is in part due to the type of data the snowpack yields and the blurred line through which a full propagation of an avalanche and a partial propagation have in the input data. The plot above shows that the actual data should be a 1 or a 0. 1 represents a partial propagation of an avalanche and 0 represents a full propagation of an avalanche. Without a partial propagation being fully defined the data gets harder to process through the neural net. It is possible to see that the neural net has learned from the data with the clusters of groupings between the 40th and last input data where the groups are more consistent.

The other reason for the network not being accurate is the network was not correctly fully trained by possibly getting stuck at a local minimum.



The training function was programmed to run up to 10000 times and this output set with a random seed shows that the training data from the Broyden-Fletcher-Goldfarb-Shanno minimization function found a minimum in only 1520 iterations. There could then be more accurate weights which would result in a better prediction from the input data.

Another possible issue is the initial weights that are set randomly and are dependent on a random seed. Different random initializations of the weights will result in different outputs and training iterations. The dimensionality of the weights in a matrix make the process of gradient descent hard to visualize and to counteract this problem with the weights the nodes could have been given certain biases to help with the predicted results.

Plot : Neural net with different random seed and 276 iterations

In the graph above the calculated and actual values are fitting more closely with the only change being the random seed. Showing that the random factor of initial weights can have a significant impact on the model as this plot was produced with only 276 iterations through the training phase. Some similarities still exist such as the likelihood not to predict a full propagation of an avalanche over a partial propagation, which could be due to these events not being exclusive enough to accurately determine the separation of them in the neural network.

Table : Learned weights from Plot 2

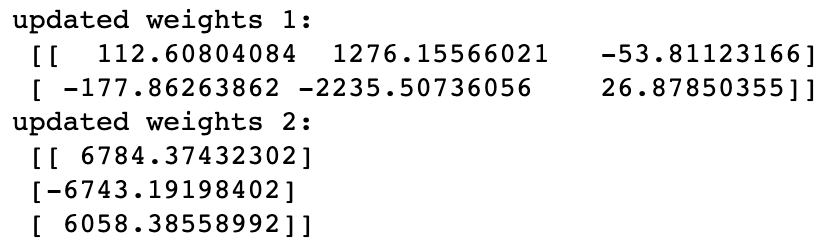
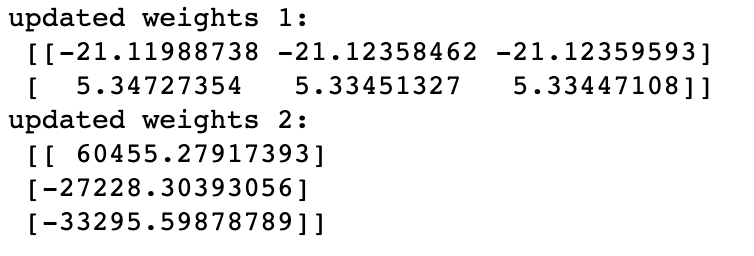


Table : Learned weights from Plot 1



The difference in calculated weights between the two random seeds is quite substantial. The outputs above are both the final learned weights of the neural nets. The weights from table 1 are the more accurate fitting plot, and the weights from table 2 are the less precise fit plot with more learning iterations shown in the first plot. Proving that no matter how many iterations the better result will come with a more global minimum finding. The learned weights also show that one set of weights is determinant of another with the weights between the input layer and hidden layer in the first table more than five times the weights of the second table which fit the model worse. Then the opposite happens for the learned weights two where the lesser fitting model in table 2 has weights that are more than ten times the better fitting weights 2 in table 1.

From the neural network and the different random seeds, the network shows that the input snowpack data might not be the most deterministic data for a neural net and instead a dataset with more independent results would be needed to calculate the propagation of an avalanche. The neural net is capable of finding a minimum where the weights can more accurately predict the propagation of avalanches but for the majority of the data points which mater which are the data points for a full avalanche is usually incorrect. Plot 2 shows that the data are grouped similarly but then is not able to determine the entire event. The problem is then not solved as picking random seeds should not be the determinant of the output data and should be able to find a better minimum with any seed. The activation function might not be accurate to this type of data as a sigmoid function is capable of intermediate values between 1 and 0, and possibly a step function could have been involved. More input data also does not aid with the results as the multiple inputs seem to overfit the model and they are not worth adding onto the neural net.

Bibliography for report and Code:

[1] 3Blue1Brown. n.d. *What Is Backpropagation Really Doing? | Deep Learning, Chapter 3*. Accessed May 1, 2019. <https://www.youtube.com/watch?v=GvQwE2OhL8I>.

[2] “Broyden-Fletcher-Goldfarb-Shanno (BFGS) Algorithm.” n.d. Accessed May 1, 2019. <https://rtmath.net/help/html/9ba786fe-9b40-47fb-a64c-c3a492812581.htm>.

[3] “Mind: How to Build a Neural Network (Part One).” n.d. Accessed May 1, 2019. <https://stevenmiller888.github.io/mind-how-to-build-a-neural-network/>.

[4] “Neural Networks Tutorial - A Pathway to Deep Learning.” 2017. *Adventures in Machine Learning* (blog). March 18, 2017. <https://adventuresinmachinelearning.com/neural-networks-tutorial/>.

[5] Stephen. (2015) 2019. *Supporting Code for Short YouTube Series Neural Networks Demystified. : Stephencwelch/Neural-Networks-Demystified*. Jupyter Notebook. <https://github.com/stephencwelch/Neural-Networks-Demystified>.

[6] “Data Set on Snow Instability - Data Set on Snow Instability - EnviDat.” n.d. Accessed May 1, 2019. <https://www.envidat.ch/dataset/data-snow-instability/resource/25cec3bc-4c79-411b-b04f-5bee43114c42>.