Lab 3: Get the Net!

ECSE 421: Embedded Systems
Department of Electrical and Computer Engineering
McGill University
Version 1.0

Adam Cavatassi and Jeremy Cooperstock

Winter 2018

1 Introduction

In this lab, you will learn the basics of a neural network inference engine. You have been provided the weights and parameters for a pre-trained neural network that will allow your myRIO board to detect one of three spatial orientations in real-time. These instructions will guide you through the basics of a neural network, the best method to import the network weights into your project, and outline the expectations for your submission. All setup and lab instructions have been verified to work with LabVIEW 2016 on the PCs in Trottier Building Room 5090.

2 Neural network preliminaries

The rest of the labs in this course as well as your final project will involve neural networks. These are trained to make predictions by being presented with examples from a labeled training set, consisting of a set of inputs, or features, and the corresponding output, over a (typically large) number of iterations. Neural networks are loosely inspired by models of biological neurons. An example of such an artificial neuron, depicted in Fig. 1, calculates the dot product of an input vector, $\mathbf{x}$ of dimension N, with a weight vector, $\mathbf{w}$ as

$$\alpha = \sum_{i=0}^{N-1} w_i x_i + b \tag{1}$$

where b is an optional bias offset.

In essence, Equation (1) determines on which side of a decision boundary, defined by the weights, the input vector lies. Such a neuron can thus be used as a linear classifier. The calculated value, α may then be passed through an activation function, Σ. For the original perceptron algorithm from 1957, this activation function is the simple step function. Other activation functions, such as the sigmoid, $\sigma(\alpha)$ and hyperbolic tangent function, $\tanh(\alpha)$, are often used, and for deep learning, a popular function is the **re**ctified **l**inear **u**nit, or $reLU$, which calculates $\text{reLU}(\alpha) = max(0, \alpha)$. These activation functions are illustrated in Fig. 2.

The activation function serves to emulate the action potential of a biological neuron. When a specific range of input values are applied, the neuron will "fire" to indicate a correlation has been detected. This

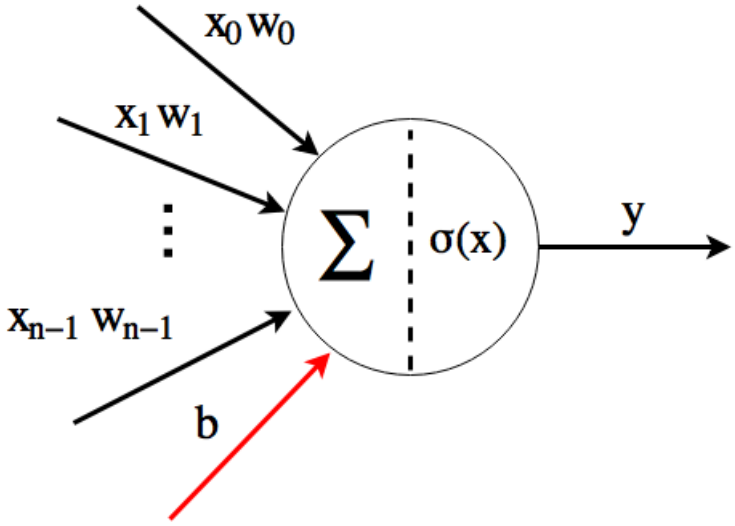

Figure 1: A single artificial neuron.

functionality is facilitated by the non-linearity.

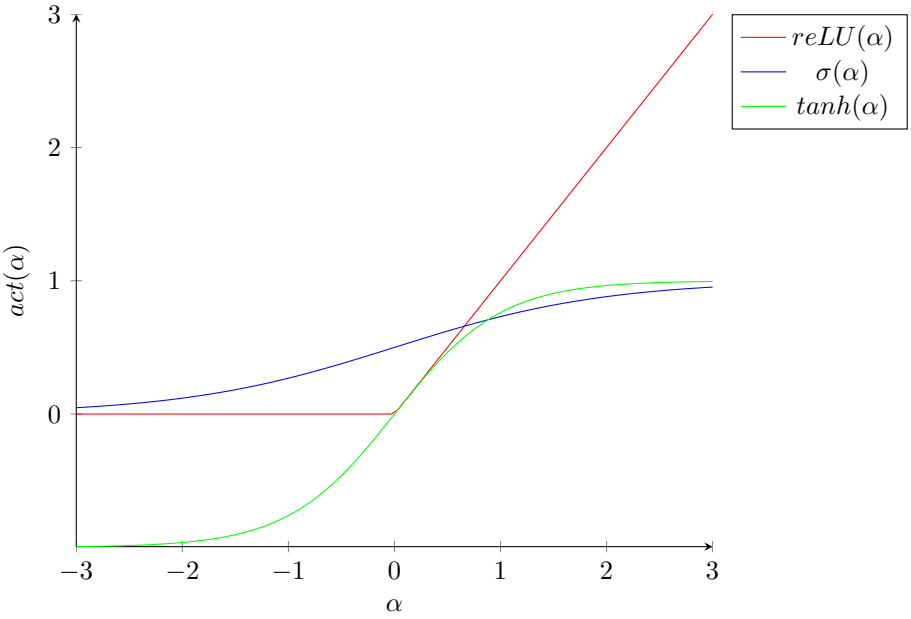

Figure 2: Plots of useful activation functions.

A single neuron can only classify a set of inputs into one of two classes. To gain more decision power, multiple neurons or *nodes* may be combined together in layers to form a multi-layer network, which allows for detection of multiple combinations of features. Nodes in subsequent layers use as inputs the activation values of nodes from previous layers. The number of layers in a network is a tunable hyper-parameter, as is the number of nodes in any given layer. The large number of these hyper-parameters can make neural networks difficult to optimize.

A neural network always has an *input layer* and an *output layer*. The input layer consists of a number of

nodes equal to the number of input features in the data. These nodes are not artificial neurons, but rather, represent the input values, directly. The number of nodes in the output layer should be equal to the number of classes the network is trying to classify. A network will typically also have one or more *hidden layers* to enable further decision-making power when working with complex data sets. A two-dimensional weight matrix represents the connections between nodes in successive layers. One full iteration of applying inputs to the first layer and receiving a prediction at the output layer is called a *forward pass*. The weights for the neural network can be learned using the *back-propagation* algorithm, which will be covered in the next lab.

Each output node is used to make a binary decision as to whether the input vector belongs to the corresponding class. As such, the output should be one-hot encoded, as seen in Table 1, for which the floating point values represent soft likelihoods that each of the classes is detected. In order to make a hard decision as to which class is being predicted, we simply take the output node with the highest value.

3 Orientation detection

The training set for this lab was generated using a myRIO board. Each training example has five inputs and one output. The inputs, $\mathbf{x}$ each consist of a single sample of the acceleration data in the x, y, and z axes, as well as the roll and pitch of the board. You will recall that you needed to collect the same data in your previous lab. The output, $o_i \in \{1, 2, 3\}$ for each example is an integer, representing one of the three possible orientations that the network is trained to recognize, as seen by the illumination of the corresponding LEDs in Fig. 3.

(a) Orientation 0 (b) Orientation 1 (c) Orientation 2

Figure 3: The three orientations of the myRIO board, with the corresponding LED illuminated.

3.1 Position-Net

The neural network for this lab, cleverly named *Position-Net*, has been pre-trained for you so you only have to implement the forward pass to create an inference engine. Position-Net consists of five input nodes, a single hidden layer with eight nodes, and an output layer of three nodes. The input nodes correspond to the five data values we know how to acquire: acceleration in the x, y, and z axes, as well as the roll and pitch of the myRIO board. The input vector $[i_0, i_1, i_2, i_3, i_4]$ is equal to the signal vector $[a_x, a_y, a_z, \theta_{roll}, \theta_{pitch}]$. The three output nodes represent the output vector $[o_0, o_1, o_2]$. This vector corresponds to the three positions

that the networks has been trained to recognize, as trained using the one-hot encoding method. The network will take a guess at which of the three positions the board is in based on the data supplied to the input nodes. For example, if the input vector $[0.001, -0.032, 0.998, 0.21, -0.018]$ were to be entered into Position-Net, you might get an output vector similar to $[0.923, 0.342, 0.098]$. In this case, the network thinks that position 0 is most likely, since o_0 has the highest value. This network has 3 layers, so there are 2 weight matrices. The activation function used in this neural network is the sigmoid function, as defined by Equation **??**. A visual representation of Position-Net can be seen in Fig. 4.

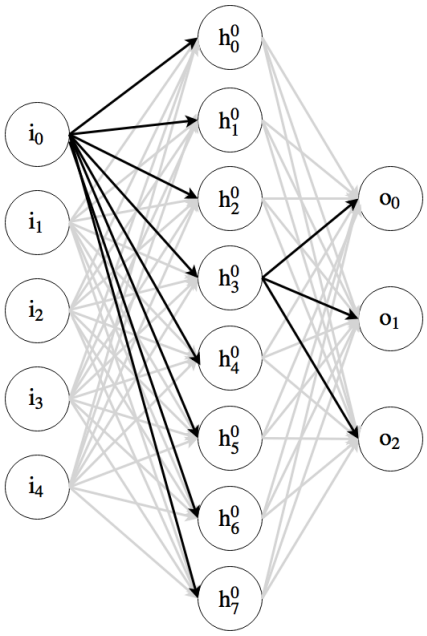

Figure 4: Position-Net, consisting of 5 input nodes, 8 hidden nodes, and 3 output nodes.

Position #	One-hot encoding
0	100
1	010
2	001

Table 1: One-hot encoding for 3 positions.

3.2 Importing the network weights into your project

To begin building Position-Net, download the .zip file called **ecse_421_lab_3_files**. Once extracted, the directory contains the LabVIEW library file **generated_data.lvlib**, the VI **load_from_file.vi**, and two 2D weight matrices stored as .csv files. The weights can be viewed in Excel. The weights connecting the input layer and the hidden layer are stored in **weights_input_h0.csv**, and the weights connecting the hidden layer and the output layer are found in **weights_h0_out.csv**.

Just as with the previous lab, start a new project in LabVIEW with your myRIO board as a target. You will need to add the library file to your project, as outlined in Fig. 5. Right-click **My Computer**, and add the library file and the VI. Drag the **generated_data** library down to the myRIO target. The library will contain a shared variable called **Weights**. A shared variable is necessary to be able to load the weight data onto the myRIO board. The myRIO has its own file system which is separate from the PC on which you are working. Because of this, saving files on the myRIO board does not allow for saving data on the PC.

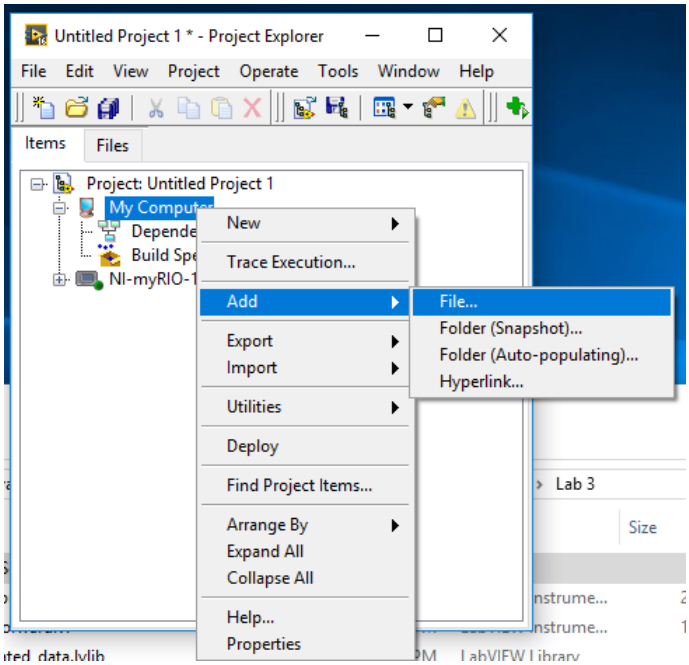

Figure 5: Add files to the PC target.

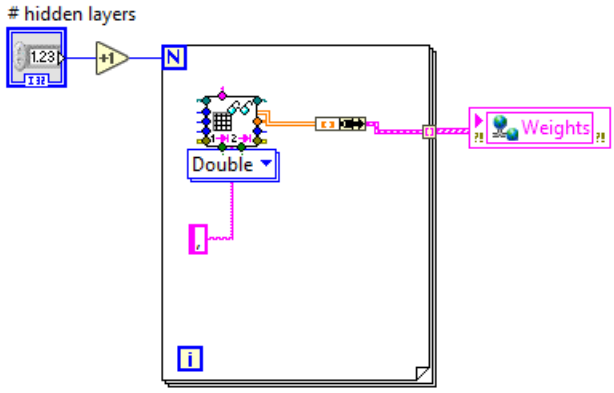

Figure 6: The VI used to import weight data from .csv files to shared variables.

At this point, your project should look the same as the one in Fig. 10. The VI used to load the weights from file should be located on the local PC target, and the library and shared variable should be located on the myRIO target. Once you have ensured that the project is properly assembled, you can run **load_from_file.vi**, seen in Fig. 6. The VI consists of a simple FOR loop to populate the shared variable. Make sure that the **hidden nodes** parameter is set to 1. If the VI is broken and not running, it may because LabVIEW did not find the shared variable at the right time. This can be fixed by adding a new shared variable reference, which can be found in the block diagram module search. If you do add a new shared variable reference, be sure to select the **Weights** variable from the drop down menu and delete the old broken reference. Once the VI does run, you will be prompted with a file selection dialog. Be sure to select **weights_input_h0.csv** first, and **weights_h0_out.csv** second. If you complete this step incorrectly, you can always re-run the VI and select the files properly. Note that every time you launch LabVIEW to work on your lab, you will need to re-run this VI to load the weights into the shared variable. The shared

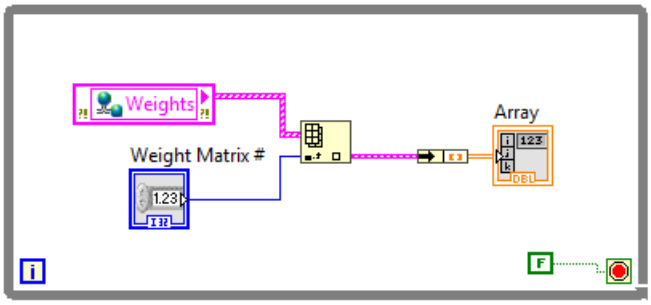

Figure 7: Build this VI to ensure the weight data is imported properly.

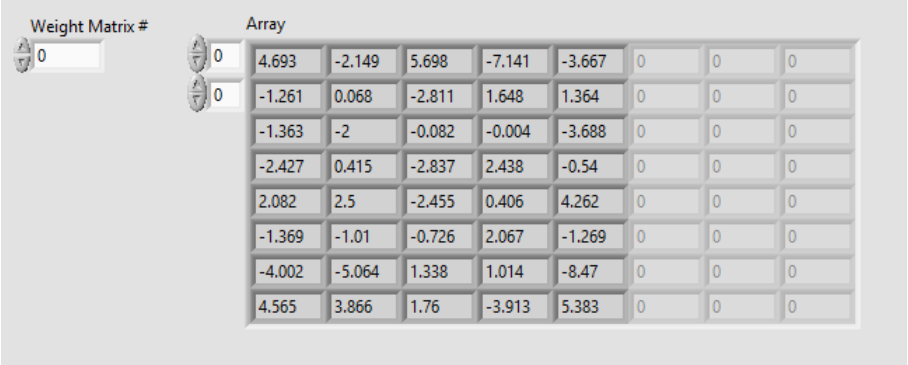

Figure 8: Ensure that the weights between the input layer and hidden layer are visible.

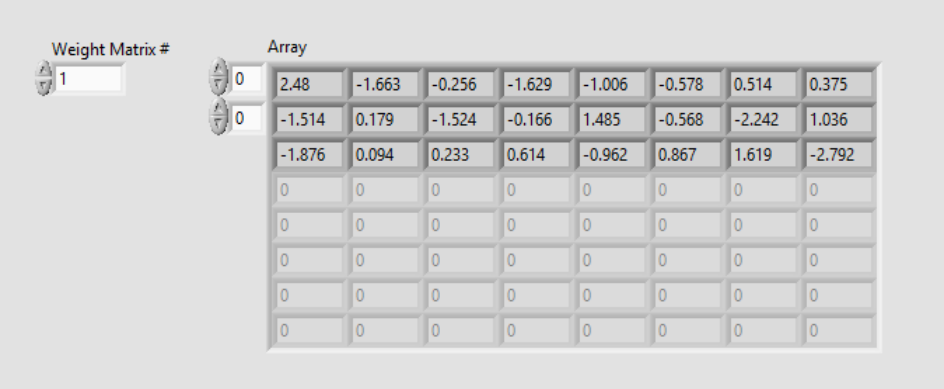

Figure 9: Ensure that the weights between the hidden layer and output layer are visible.

variable will *not* retain the stored values after you close LabVIEW.

Once you have successfully run the **load_from_file** VI, you should make a test VI on your myRIO board to verify that the values are indeed stored in the shared variable. Build a VI that looks like the one in Fig. 7. The format of the shared variable is a 1D array of clusters. Clusters are a LabVIEW container that can be custom-built. In this case, the format of the clusters is a 2D array of doubles. Note that you need to unbundle a cluster in order to access its contents. In summary: the shared variable **Weights** is a 1D array of clusters of 2D arrays of double. The reason the weights are not in a more straightforward 3D array is because LabVIEW does not support jagged arrays. For a neural network, each 2D array of weights between layers will usually have dimensions of different sizes. Once your test VI is working, you should be able to view the contents of the weight arrays, as shown in Figs. 8 and 9. Verify the values of the arrays with the

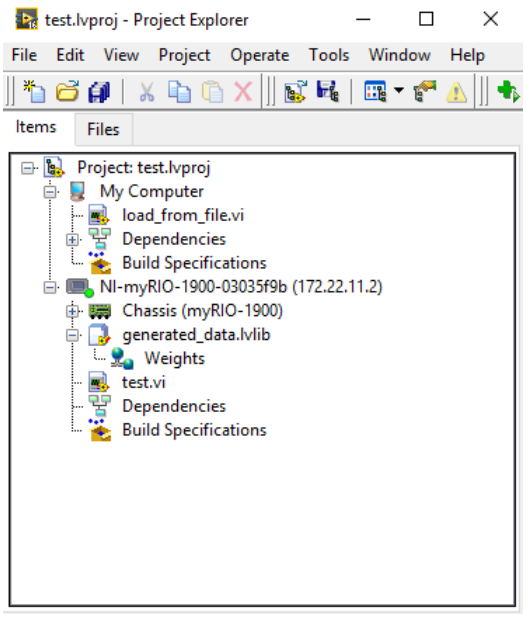

Figure 10: Your project explorer should look like this after adding all of the supplied files.

.csv files in Excel. Note that the first set of weights is an 8×5 matrix, and the second set of weights is a 3×8 matrix.

Now that you can see that the weights are properly stored in the shared variable, you can use them to build your feed forward inference engine by implementing Position-Net. Although Position-Net is a relatively small and simple neural network, it is important to note that a highly reconfigurable design will be very useful for future labs and projects. Hand-wiring the network together without any looping or modular configuration will make your work in this lab not very useful for future work.

4 Submission requirements

For this lab, you will be required to submit a .zip file containing your project file, all VI files that are part of your project, and screenshots of each the final block diagram and final front panel for all VIs. You will also be required to submit a URL to a private YouTube video with a maximum length of 120 seconds which demonstrates the full functionality of the inference engine in real-time. Show that your inputs are your 5 signals from the previous lab, and your outputs are LED0, LED1, and LED2 on the myRIO board. Show that each position is clearly recognized by the neural network, as indicated by the LEDs.