

Writing Neural Network Code: Introduction to TensorFlow, Hands-On

Class 2: Neural Network Simulation and Programming

May 12, 2020

Charles J. Lord, PE
President, Consultant, Trainer
Blue Ridge Advanced Design and Automation

This Week's Agenda

- 5/11 A Brief History of Artificial Neural Networks
- 5/12 Neural Network Simulation and Programming
- 5/13 TensorFlow Hands-On Part 1: Hello World!
- 5/14 TensorFlow Hands-On Part 2: Defining and Building Your Network
- 5/15 TensorFlow Hands-On Part 3: Teaching and Testing and Conclusion

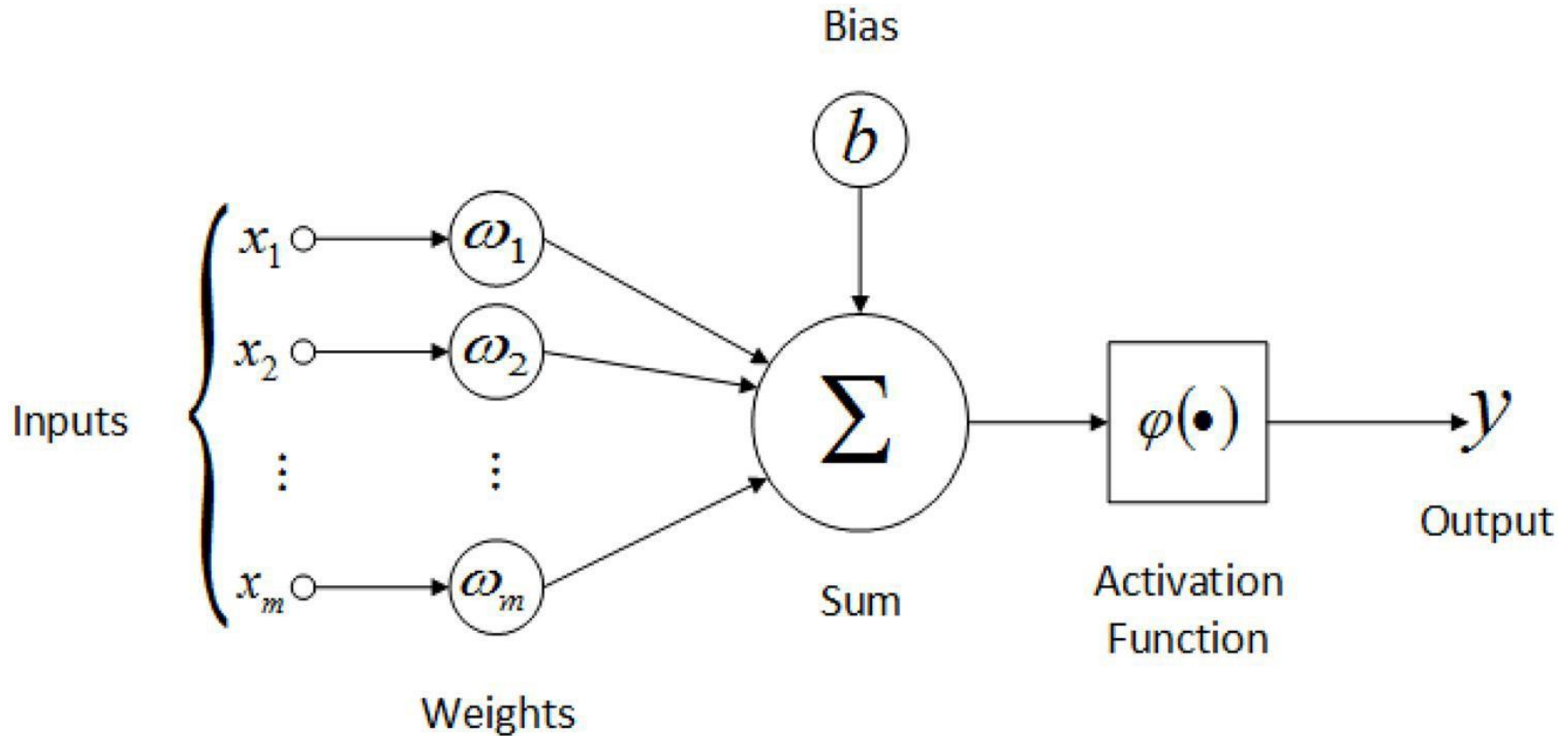
This Week's Agenda

- 5/11 A Brief History of Artificial Neural Networks
- 5/12 **Neural Network Simulation and Programming**
- 5/13 TensorFlow Hands-On Part 1: Hello World!
- 5/14 TensorFlow Hands-On Part 2: Defining and Building Your Network
- 5/15 TensorFlow Hands-On Part 3: Teaching and Testing and Conclusion

From Yesterday

- We got a jump on today's class by quickly going over some of the historical programming systems yesterday
 - Torch
 - Keras
 - TensorFlow
- Today we will look at the mathematics of a simple neural network.

Basic Neuron Model



Question 1 – What is the activation function of a biological neuron?

Our example (trivial)

4 input nodes

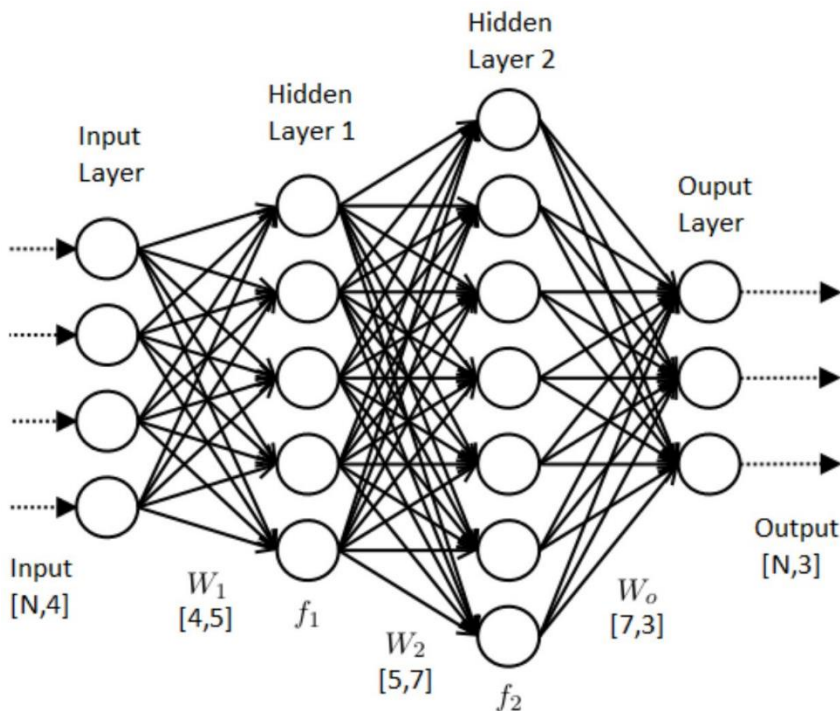
5 hidden 1 nodes

7 hidden 2 nodes

3 output nodes

Takes in 4 different values

Gives 3 output values
(probabilities)



Weight and Bias

- In the biological neuron (BN) the learning that takes place is thought to be a chemical reaction within the cell that gives different sensitivities to each input as well as changing the firing threshold of the output (pulse or step function)
- In the Perceptron, the neurons were made with the same structure – a binary output

$$f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0, \\ 0 & \text{otherwise} \end{cases}$$

Weight and Bias - 2










- Another weakness of the Perceptron was the massive bank of potentiometers
- Each of these set a weight for each input at each node as well as bias!
- Programming was extremely long and tedious – trial and error
- No surprise that the device fell out of favor fairly quickly



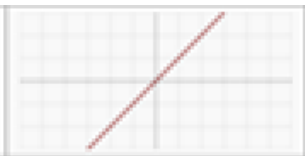
The More the Merrier

- The human brain can be so powerful with limited neuron output due to the mass scale of neurons. Approximately 83 billion.
- It was recognized early that an analog function of some sort would allow perception with far fewer neurons.
- This led to the various activation functions we looked at yesterday

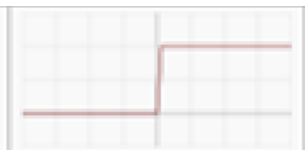
Activation Functions

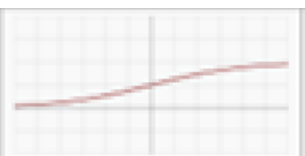
Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

Activation Functions 2

Identity		$f(x) = x$	$f'(x) = 1$
----------	---	------------	-------------

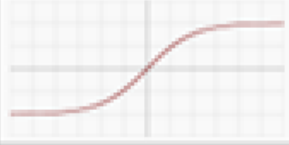
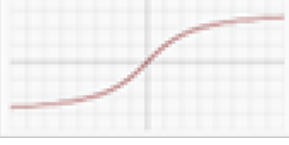
The Identity function was an early favorite, primarily due to its simplicity. However, the negative values made it unstable when trying to tune and would often not converge on a solution

Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
-------------	---	--	---


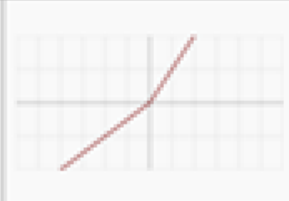
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
----------------------------	--	-------------------------------	--------------------------

The log or soft step function eliminated the negative outputs of the identity function but the positive value at the origin made it difficult to converge for some applications

Activation Functions 3

Tanh		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$

The hyperbolic tangent and arctangent functions are best for some applications but can still have convergence issues for general purpose applications

Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$

The ReLU function eliminated the negative outputs of the identity function and is the most popular AF but the PReLU is better suited for some apps

What's Best

- Experimentation and experience are the best teachers for the best activation function to use in a CNN. All nodes in a layer should use the same AF but different layers can have different functions.
- ReLU, being the most popular for general purposes, is a good starting point but a good tuner should try others, particularly if ReLU is slow to converge

The Math of a CNN

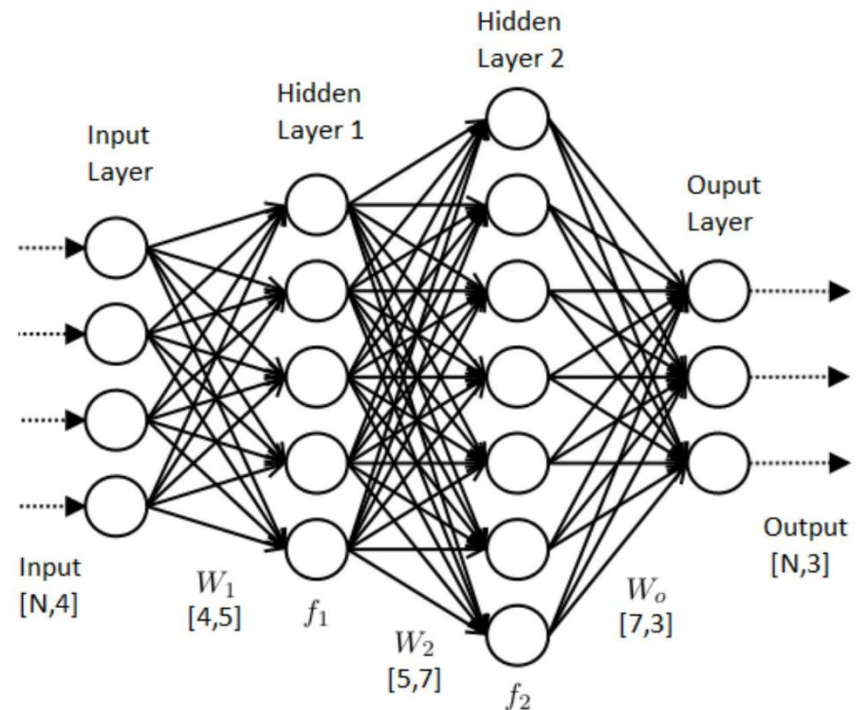
- As we saw before, the output of a node m with n inputs is

$$y_m = \phi \cdot \sum_{i=0}^n w_{mi} x_{mi} + b_m$$

- Where ϕ is the activation function
- w_{mi} is the weight for each input
- x_{mi} is each input
- b_m is the bias for that node

Recursiveness

- For the oversimplified CNN in our example, the math becomes a bit unwieldy but can be calculated – it is a simple task of nesting the outputs of preceding nodes as the inputs to the next node.



Question 2 – What is an application where we may only have 4 in and 3 out?

Making Sense of the Result

- As the most common use of a CNN is to perceive, recognize, or classify something, the output of the output layer nodes is usually converted to a value from 0 to 1, interpreted as a probability that the value or condition reported by that node reflects the value or condition at the inputs. For example, in our ultimate example of the MNIST dataset, we will have 10 outputs that each reflect the probability that an image is of a particular numeric digit.

Output Example

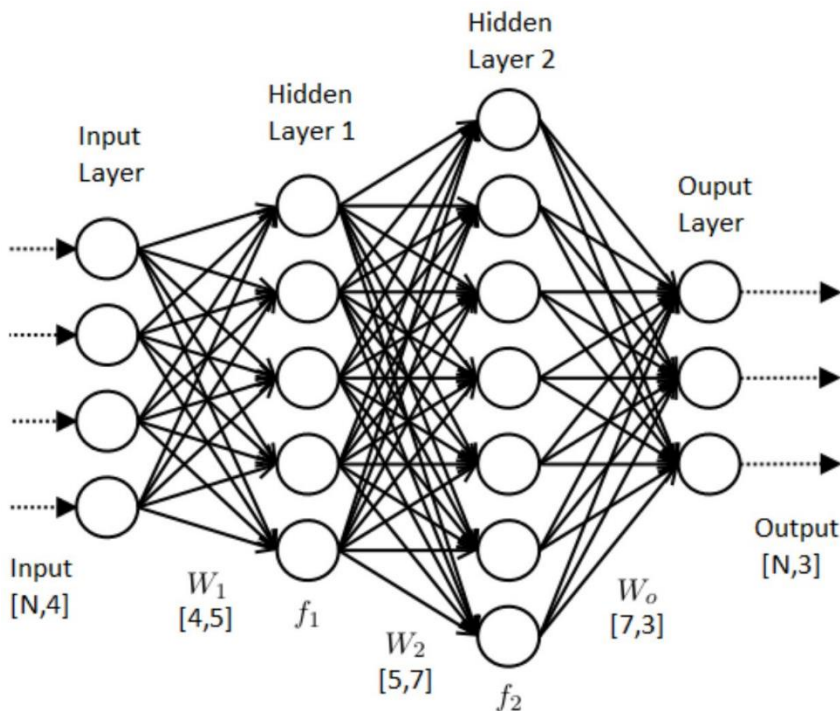
1. 7.57983543e-09
2. 1.50700696e-09
3. 6.43297608e-06
4. 1.38727250e-04
5. 9.80789343e-13
6. 1.31976194e-07
7. 6.88661639e-14
8. 9.99849916e-01
9. 1.18184182e-07
10. 4.69114502e-06

- Note that all values are much less than 1, except for the 9th value
- This example output shows that the CNN is saying there is a .000076% probability that the first condition is met, while there is a 99.985% chance that the 8th condition is true

Back to the Pots

- The potentiometers in the Perceptron were its downfall; this was still a stumbling block until Paul Werbos demonstrated the use of back propagation to program a CNN
- The concept of reverse propagation is simple in theory but extremely complex in practice

Too Many Variables



- For our 4|5|7|3 network there are
 - 73 input weights
 - 19 biasesFor a total of 92 variables in our tweaking or training
- We need a method to determine if we are converging on a solution

Regression and Loss

Simple polynomial data fitting and regression have used the sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

And we use this to fit to a one degree polynomial $mx + b$ by stating that the probability of a point being in one condition as $\sigma(mx + b)$ and the inverse as $1 - \sigma(mx + b)$

Classification

Further defining $\sigma(mx + b)$ as $h(x)$ (for simplicity) we can define the likelihood of a condition y_i (as opposed to any other condition y_0) as

$$L(y_i) = h(x)^{y_i} [1 - h(x)]^{(1-y_i)}$$

This will give a value from 0 to 1 as to how close $h(x)$ fits the model (ideal system or hypothesis)

Loss

Statisticians, looking to maximize the likelihood of matching the system, take the logarithm of the likelihood to what is referred to as the loss

$$loss = -y_i \log(h(x)) - (1 - y_i) \log(1 - h(x))$$

But what if we have MULTIPLE variables that we need to optimize for minimum loss?

Softmax

There is an operation that takes the sigmoid into an array of values – and returns an array.

For an array of softmax function terms denoted by $\sigma(x)_j$ with N terms, softmax of x_j can be found by

$$\sigma(x)_j = \frac{e^{x_j}}{\sum_{i=0}^{N-1} e^{x_i}}$$

Putting It Together

- Softmax and loss are complex computations, but tools like TensorFlow has these functions built-in. We will put this in practice at the end of the week when we train and optimize our neural network.

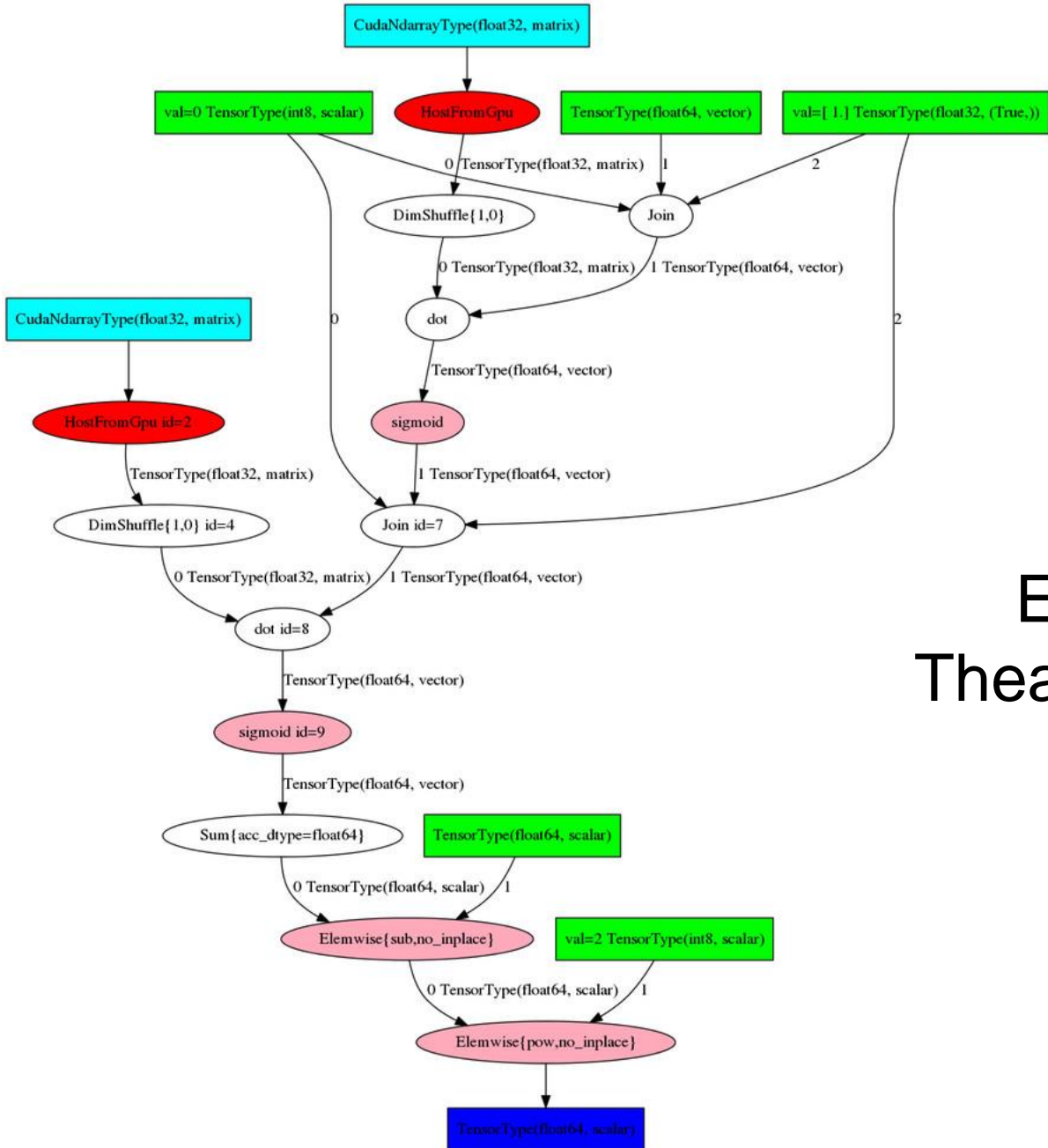
Handling The Data

- As we briefly covered yesterday, various components from previous tools get re-used in subsequent tools for designing, building, training, and testing neural networks
- Torch, the first of the 'modern' tools developed in 2002, was designed for work in complex vector and matrix calculations. It took the concept of **tensors** from advanced mathematics (and used in fields and mechanics) to be a container for any combination of scalars, vectors, and multi-dimensional arrays

Graphs

- One of the legacies of the Theano toolset was the concept of the graph. A graph is a collection of data and operations in one package. It got its name from the fact that there were visualization tools that would map out a visual representation of what the package contained. And by using naming conventions, a graph could be selectively executed as needed.

Question 3 – What is the visualization tool in TensorFlow?



Example Theanos Graph

Tomorrow!

What do you get when you put together
Tensors + Graphs + Google?

(hint: it goes with the flow)

This Week's Agenda

- 5/11 A Brief History of Artificial Neural Networks
- 5/12 Neural Network Simulation and Programming
- 5/13 TensorFlow Hands-On Part 1: Hello World!
- 5/14 TensorFlow Hands-On Part 2: Defining and Building Your Network
- 5/15 TensorFlow Hands-On Part 3: Teaching and Testing and Conclusion

Please stick around as I answer your questions!

- Please give me a moment to scroll back through the chat window to find your questions
- I will stay on chat as long as it takes to answer!
- I am available to answer simple questions or to consult (or offer in-house training for your company)

c.j.lord@ieee.org

<http://www.blueridgetechnc.com>

<http://www.linkedin.com/in/charleslord>

Twitter: @charleslord

<https://www.github.com/bradatrainning>