Deep Learning With TensorFlow

An Introduction To Artificial Neural Networks

By Brian Pugh

CMU Crash Course 1/28 2017

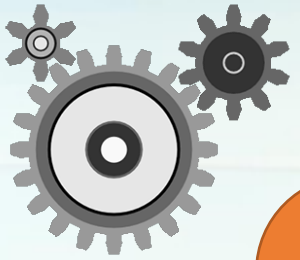
Goals

- What is Deep Learning?
- What is an Artificial Neural Network?
- A Basic Artificial Neural Network (Feed Forward Fully Connected)
- Implement a basic network for classifying handwritten digits

Deep Learning

- A branch of Machine Learning
- Multiple levels of representation and abstraction
- One step closer to true “Artificial Intelligence”
- Typically refers to Artificial Neural Networks
- Externally can be thought of as a black box
- Maps inputs to outputs from rules it learns from training
- Training comes from known labeled input/output datasets.

Cat/Dog Classifier



DEEP LEARNING

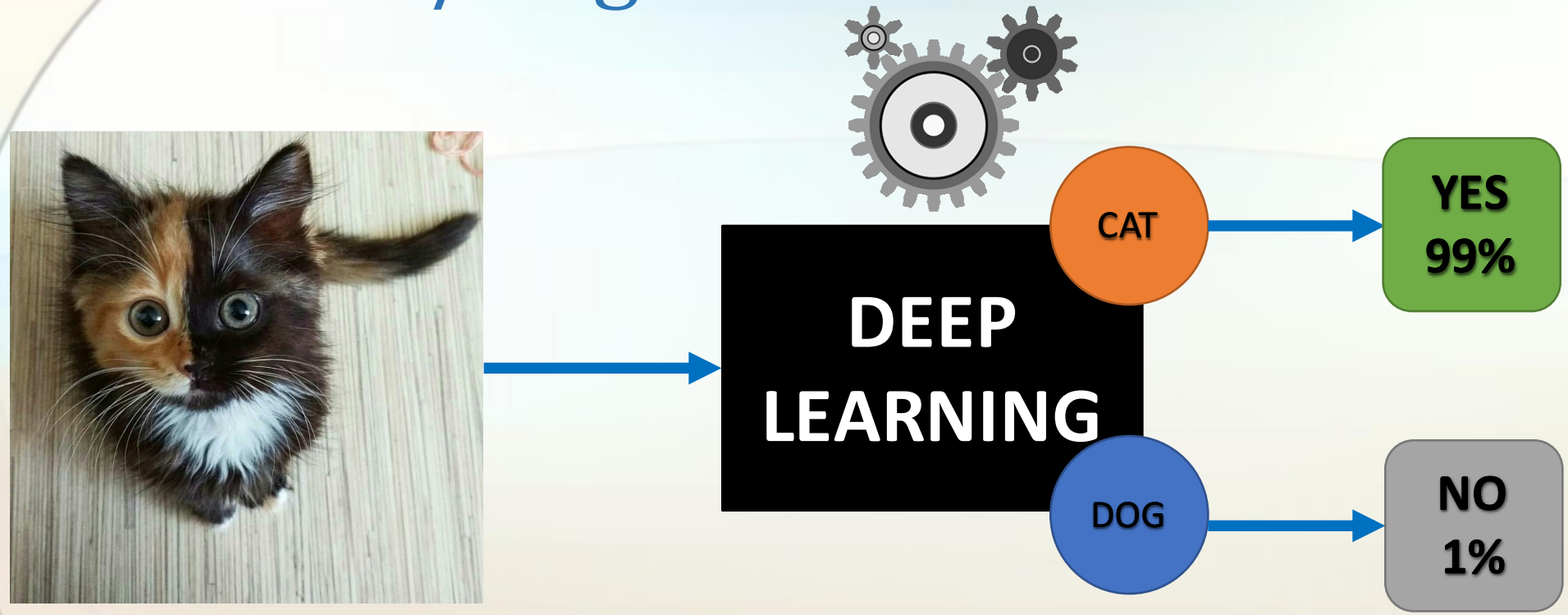
CAT

DOG

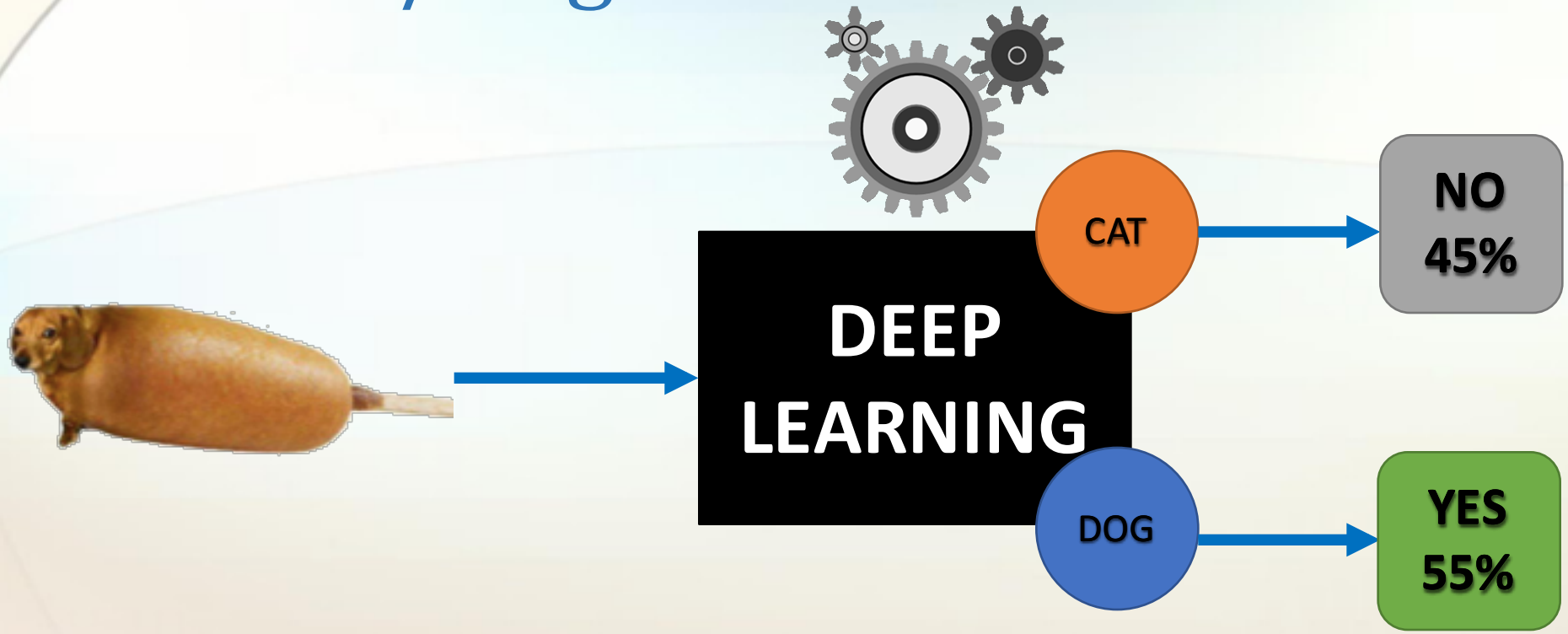
NO
1%

YES
99%

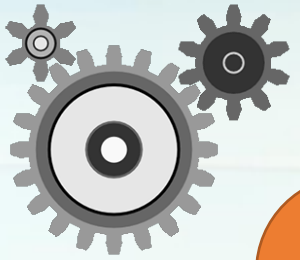
Cat/Dog Classifier



Cat/Dog Classifier



Cat/Dog Classifier



DEEP LEARNING

CAT

DOG

YES
55%

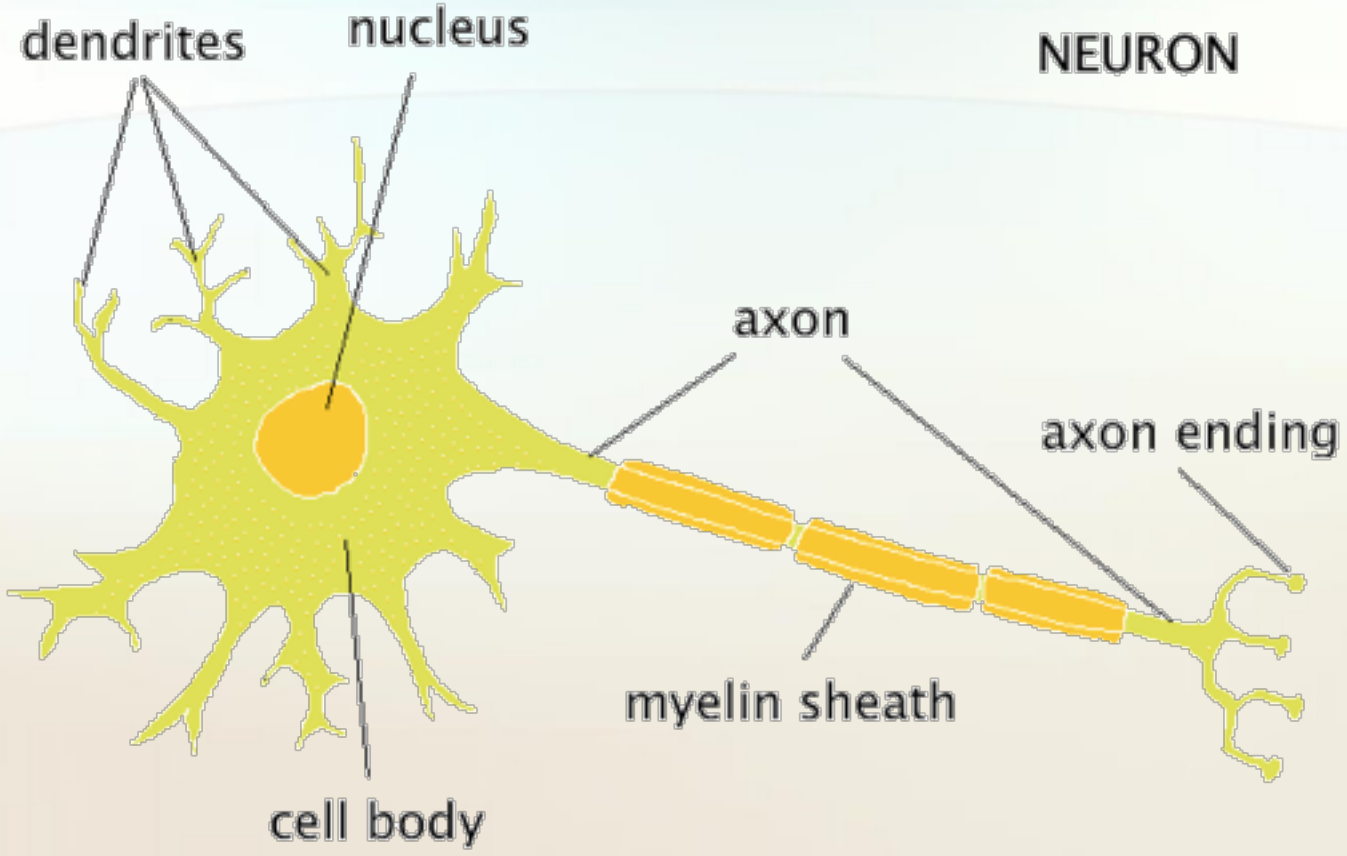
NO
45%

Cat/Dog Classifier

Topic Ordering (Logistics)

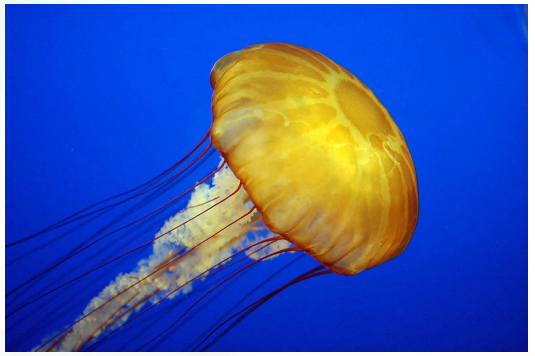
- There are many components in Artificial Neural Networks
- They all come together to make something useful.
- If something is not clear, please ask!
- Hard to figure out the best ordering of topics.

Neuron (Inspiration)



Source: http://webspace.ship.edu/cgboer/neuron.gif

Number of Neurons



Animal	Number of Neurons
Common Jellyfish	5,600
Ant	250,000
Frog	16,000,000
Cat	760,000,000
Humans	86,000,000,000
African Elephant	257,000,000,000

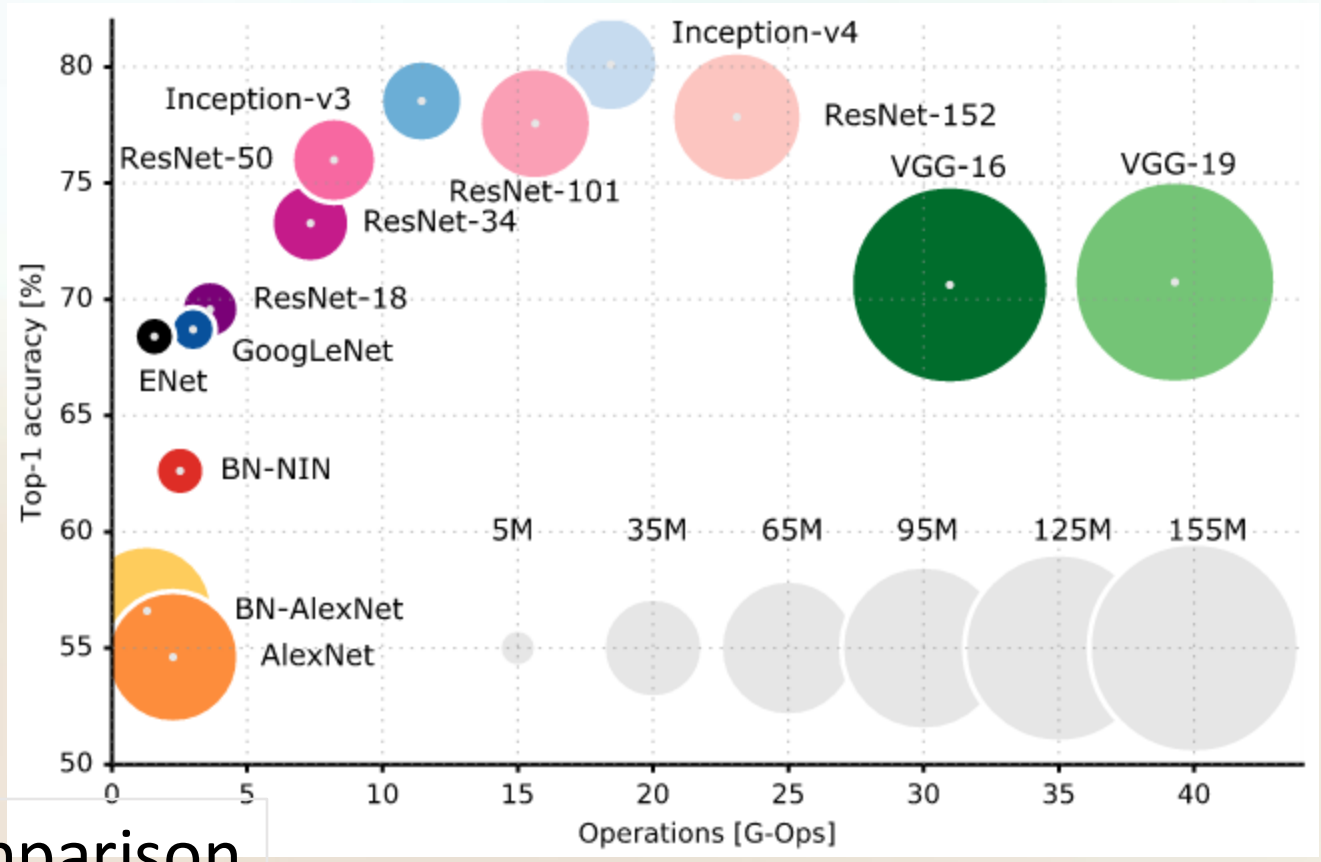
*Not a direct comparison

Source: https://en.wikipedia.org/wiki/List_of_animals_by_number_of_neurons

Current Network Architectures

- Blob Size = Number of Parameters (connections between neurons)

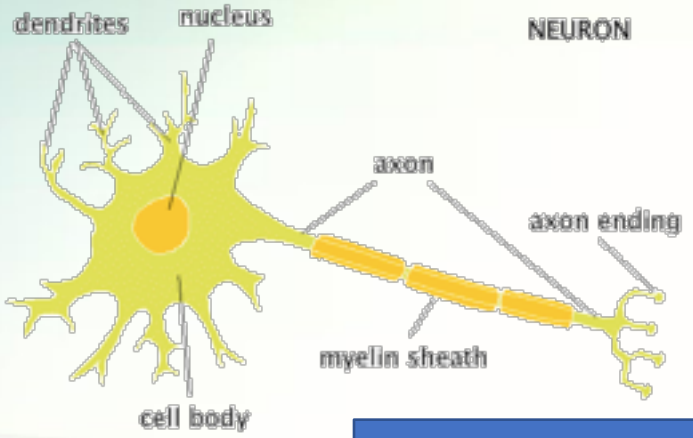
ResNet-152 has
167,552 Neurons



*Not a direct comparison

Source: Deep Residual Learning For Image Recognition by Kaiming He et. al.

Artificial Neuron



Some scalar values

Some scalar values (initialized randomly)

$$relu([x_1 \quad x_2 \quad x_3]\begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix})$$

May use some output decision function

Input x_1

Input x_2

Input x_3

w_1

w_2

w_3

Sensory Input or Axons of other neurons

Dendrites

SUM

Nucleus

Some scalar value

Axon

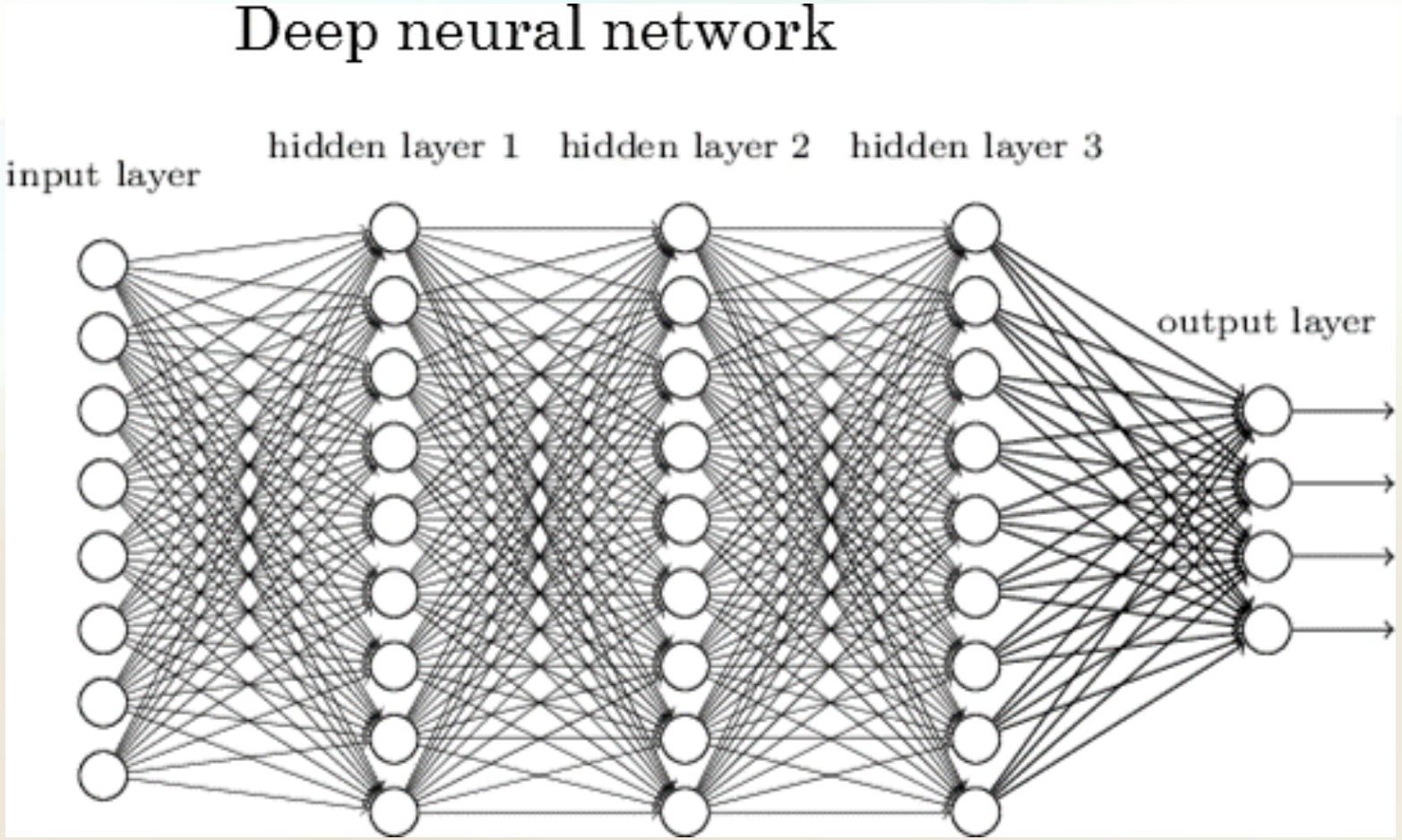
Some scalar value

Nonlinear Activation Function

$$RELU(x) = \begin{cases} 0 \text{ if } x < 0 \\ x \text{ if } x >= 0 \end{cases}$$

Axon Ending (possibly to other neurons!)

Combining Neurons Into Layers



Backpropagation (How the network learns)

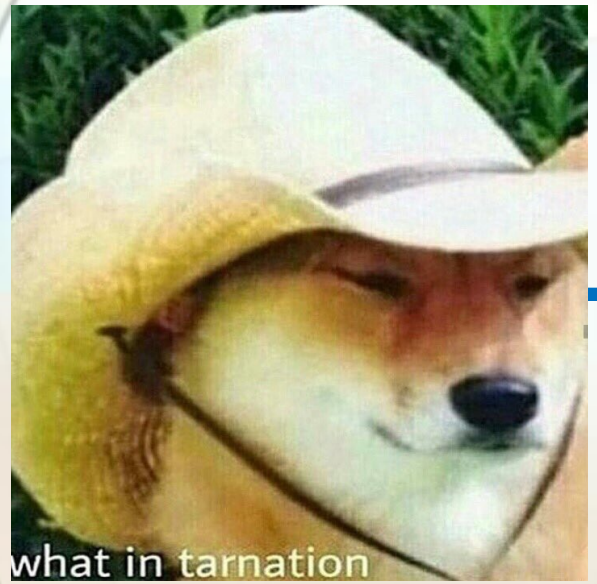
- How the network learns useful weights
- Will not go into depth on how it works
- **Don't** need to know it to use off-the-shelf components in TensorFlow
- **Do** need to know if you want to implement custom layers

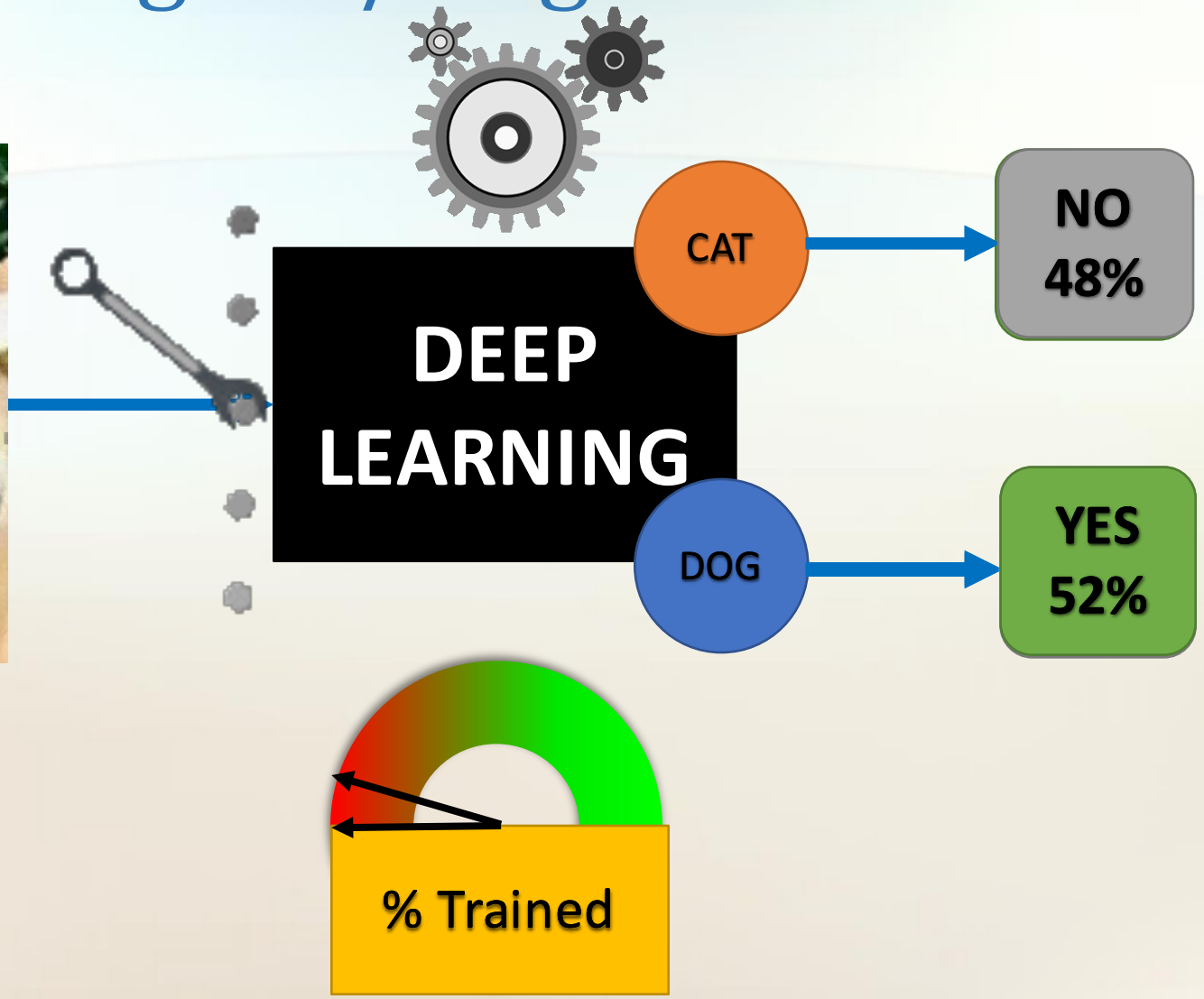
- Basically through continuous differentiation (calculus), the network figures out how much each parameter (weights) contributed to an error and tweaks it to reduce the error.

Training In A Nutshell

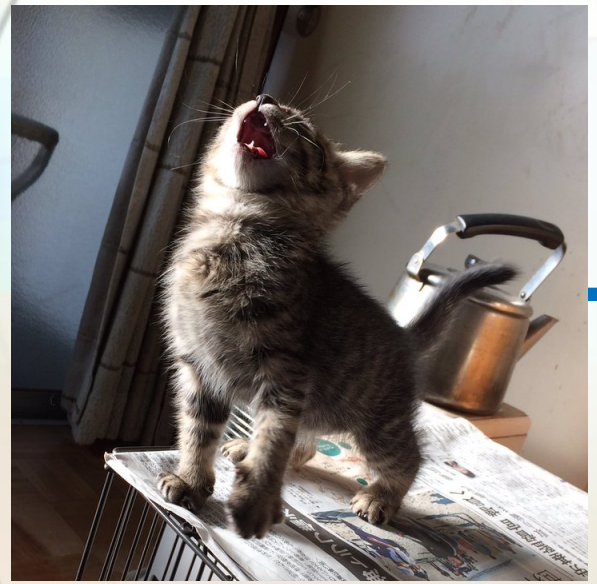
1. Forward pass some example input
2. Compare the network output with what the output **should** be
3. Backpropagation back through the network
4. Update weight values via Backpropagation
5. Now, whenever the network gets that input, the output should be closer to the goal output.

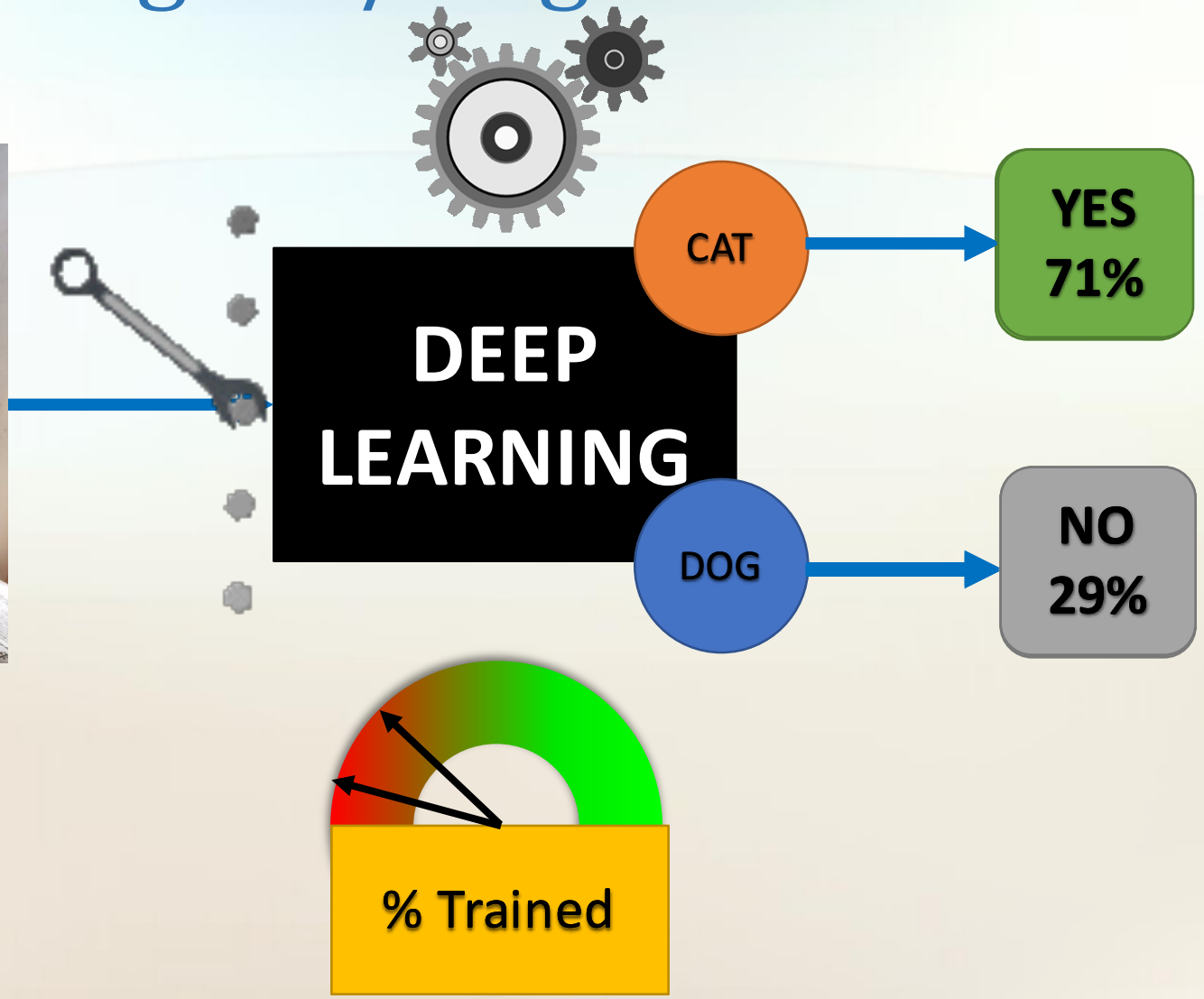
Training Cat/Dog Classifier



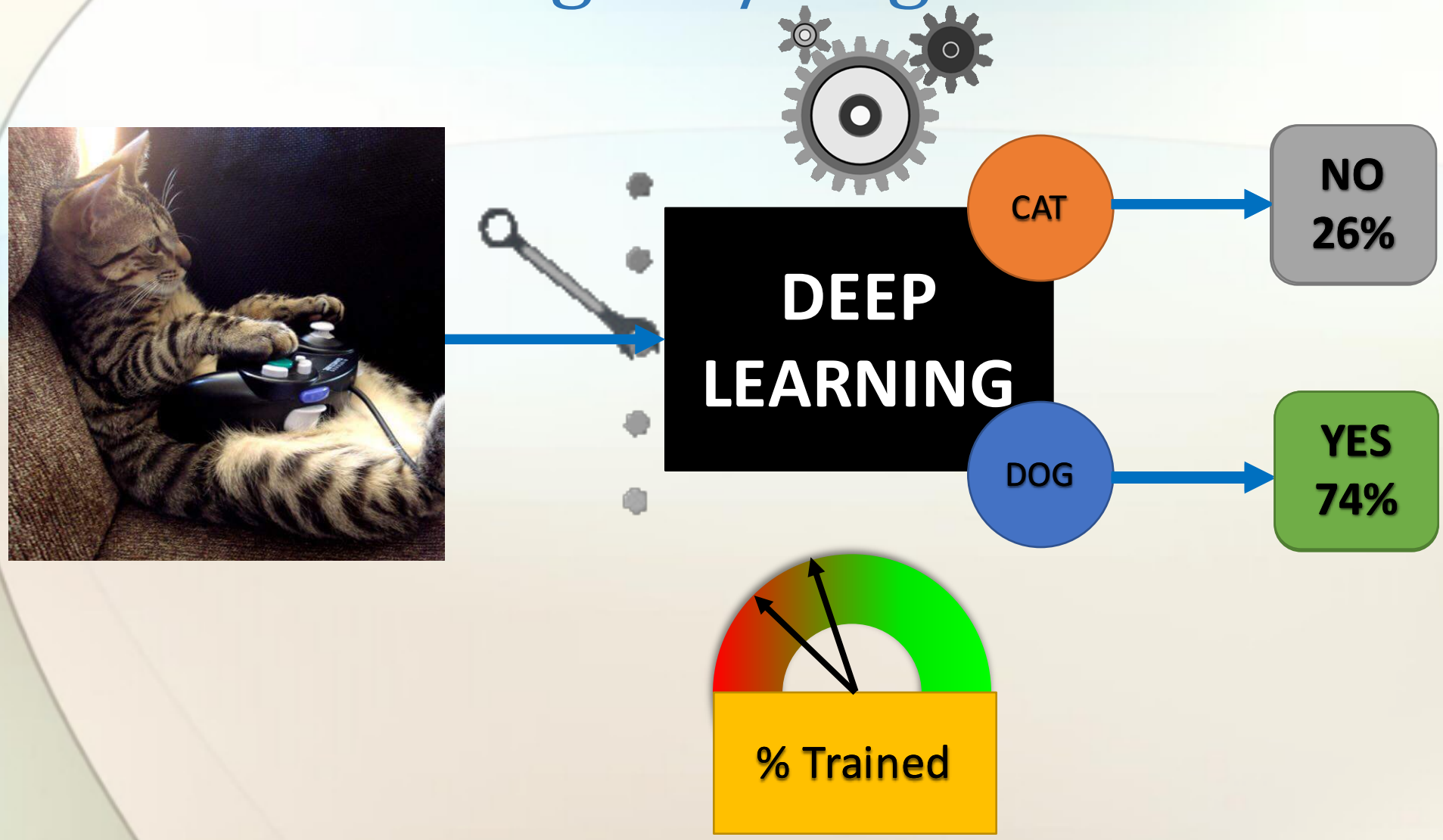


Training Cat/Dog Classifier

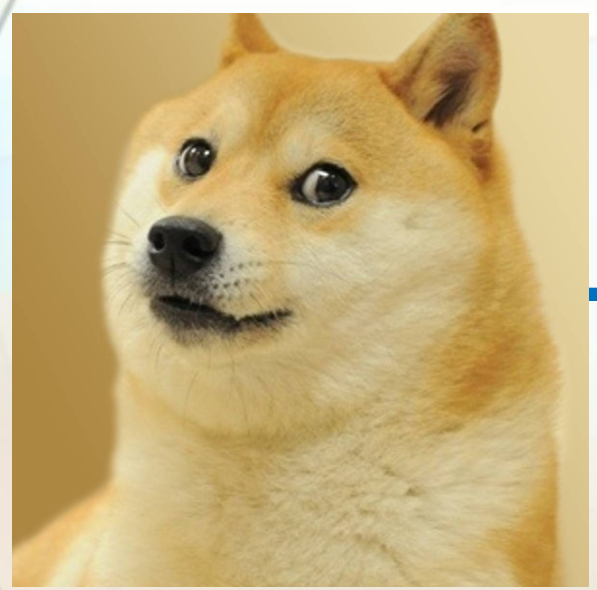


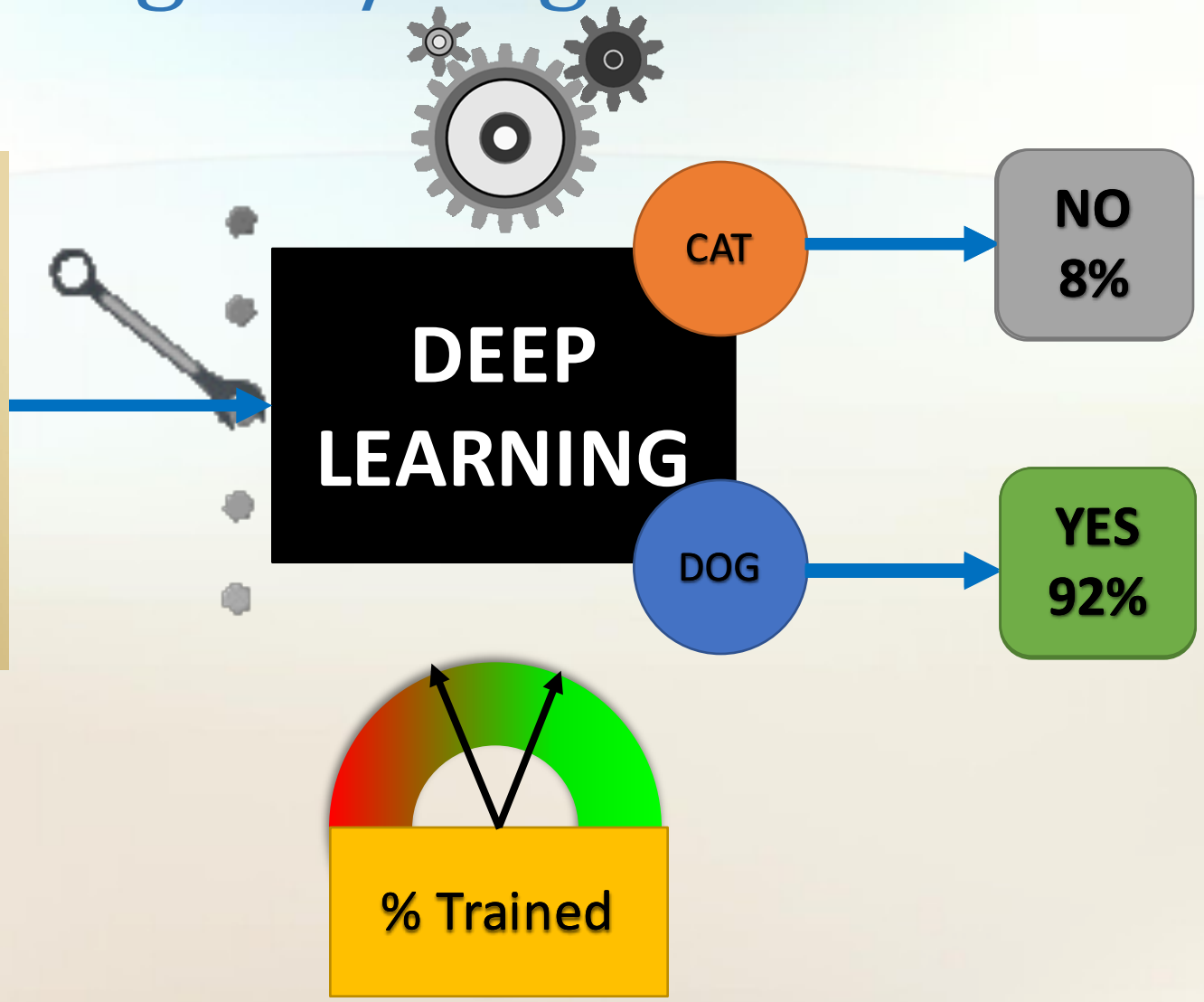


Training Cat/Dog Classifier



Training Cat/Dog Classifier





Training Cat/Dog Classifier

DO THIS TENS/HUNDREDS OF THOUSANDS OF TIMES

GPUs! (Aside/Optional)

- Each neuron can be computed in parallel (independent)
- GPUs have hundreds (even thousands) of relatively weak cores
- Nvidia has a virtual monopoly (thanks to the CUDA Toolkit)
- Nvidia supplies AWS, Azure, etc.
- Google signed a deal with AMD (but also uses Nvidia)

TensorFlow!

- Google's publicly available Deep Learning library
- In competition with Caffe, Torch, etc
- Is becoming more and more popular in industry.

- Normally we leverage the power of GPUs (thousands of time speed up), but the installation for TensorFlow is Linux-only and more tedious than CPU-only installation.

Installation

- Download and Install Anaconda (Python 3.5)
- Download and install TensorFlow in a conda environment called "tensorflow"
- Download and install the Spyder IDE within the conda environment:

```
conda install -n tensorflow spyder
```

Importing the Library

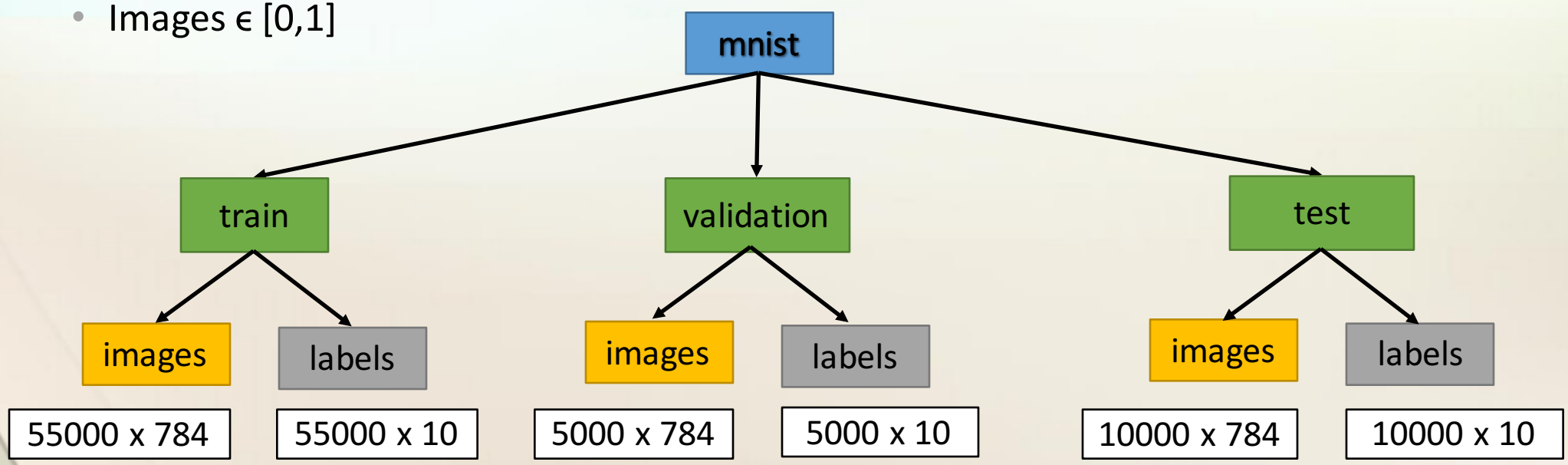
```
import tensorflow as tf
```

- Tensorflow functions can now be called using the tf prefix. Example: tf.session()

Importing the MNIST Dataset

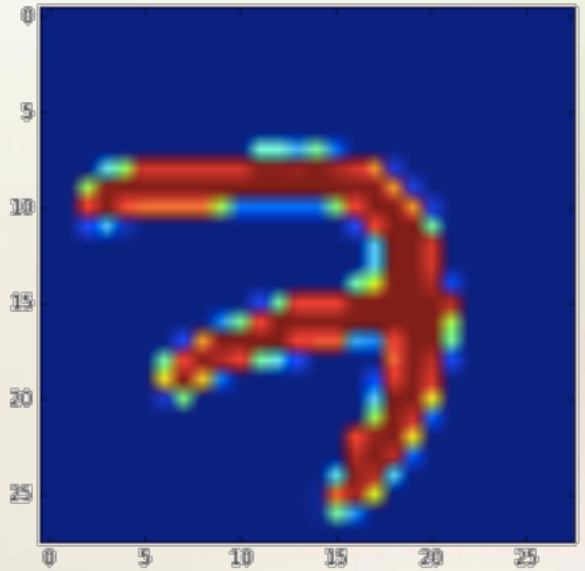
```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

- This downloads and loads in the MNIST digits dataset.
- Note: 784 = 28 x 28
- Images ϵ [0,1]



Viewing One MNIST Image

```
import matplotlib.pyplot as plt
im = mnist.train.images[0,:]
label = mnist.train.labels[0,:]
im = im.reshape([28,28])
```



Softmax and One-Hot Encoding

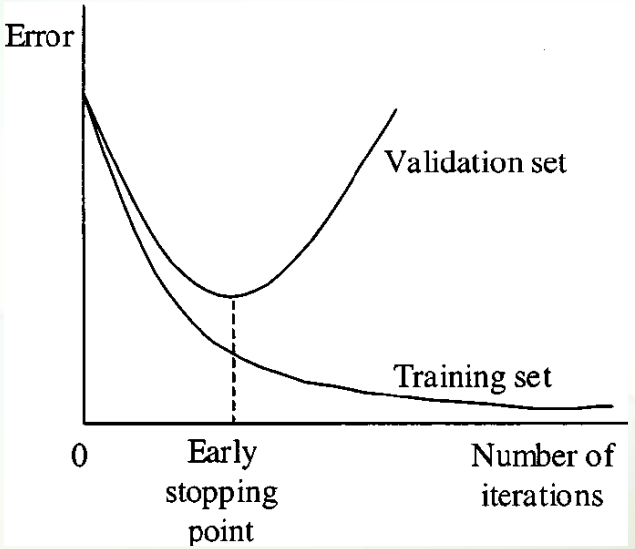
- We want the network to output a percent certainty it believes some image belongs to some label.
- Softmax remaps the output layers to percentages.

$$softmax(v_j) = \frac{e^{v_j}}{\sum_i e^{v_i}}$$

- Example: [1.0, 1.0, 6.0] -> [0.0066, 0.0066, 0.9867]

- One-Hot Encoding (with 10 options): 3 -> [0 0 0 1 0 0 0 0 0 0]

Train, Validation, Test



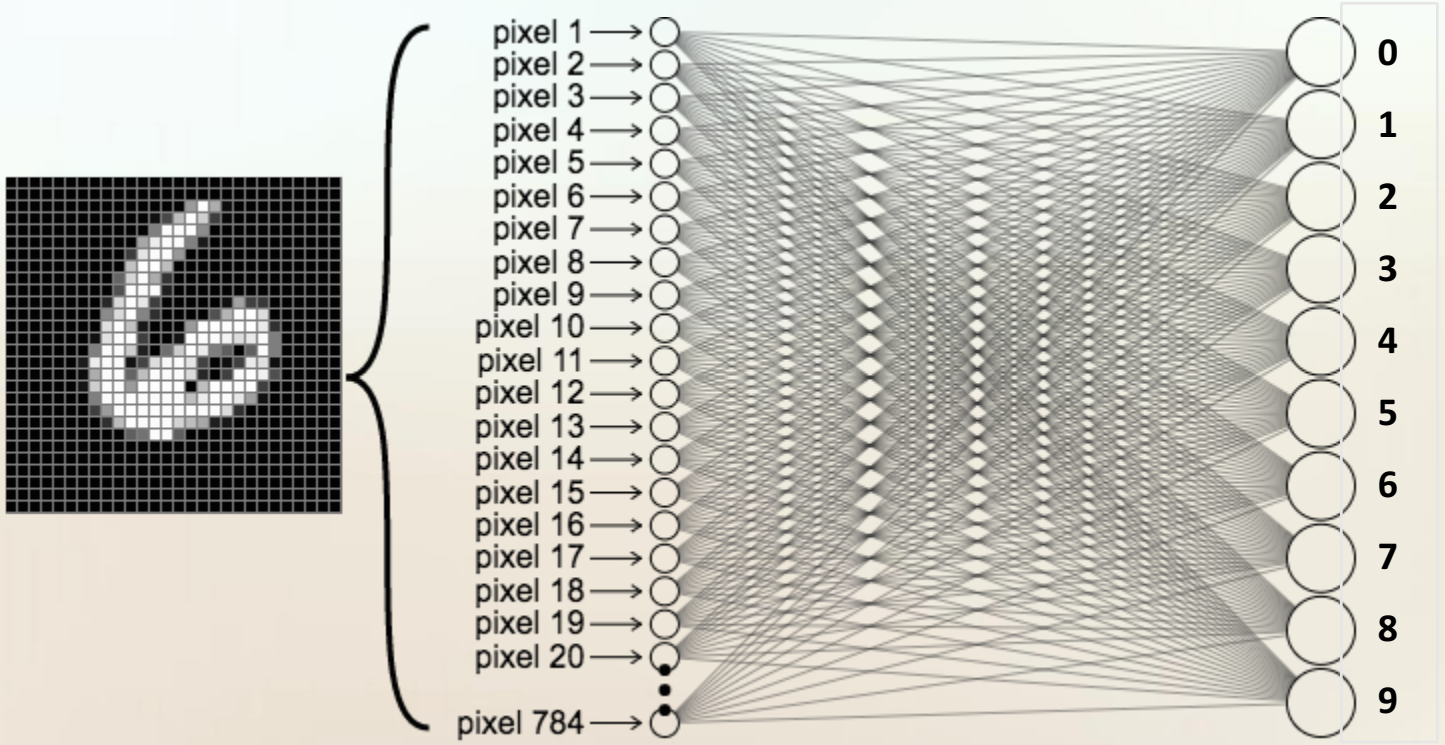
- Typical Data Breakup:

	Train	Validation	Test
Percentage:	80~90	5~10	7~15

- Important that every dataset is representative
- Train on the training dataset.
- Check performance during training with Validation dataset
- See actual network performance with Test dataset.

Define Model (Not So Deep Learning)

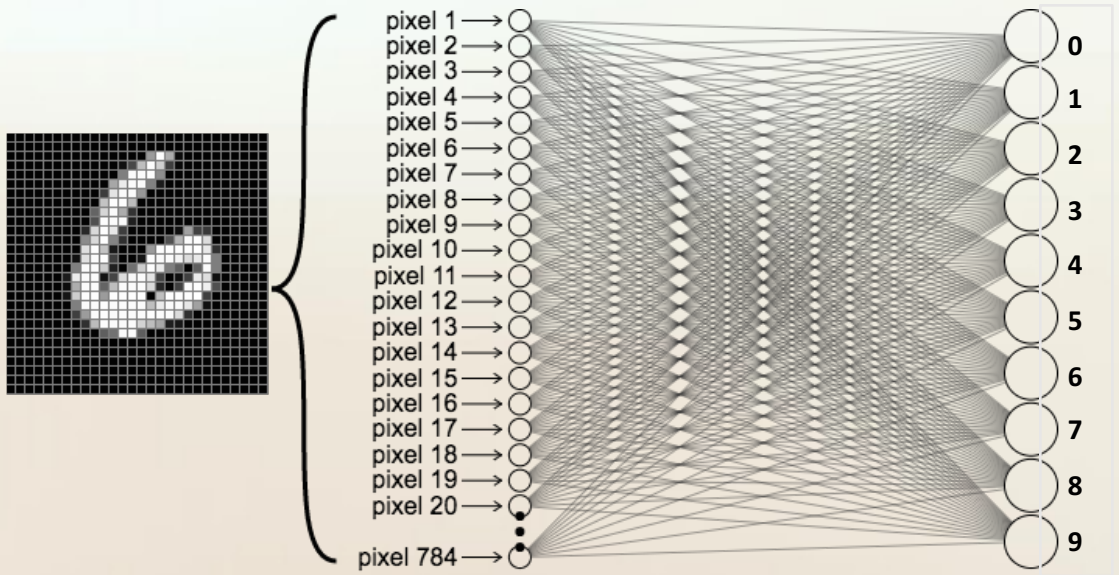
- We will construct a feedforward fully connected neural network that is 1 layer deep.



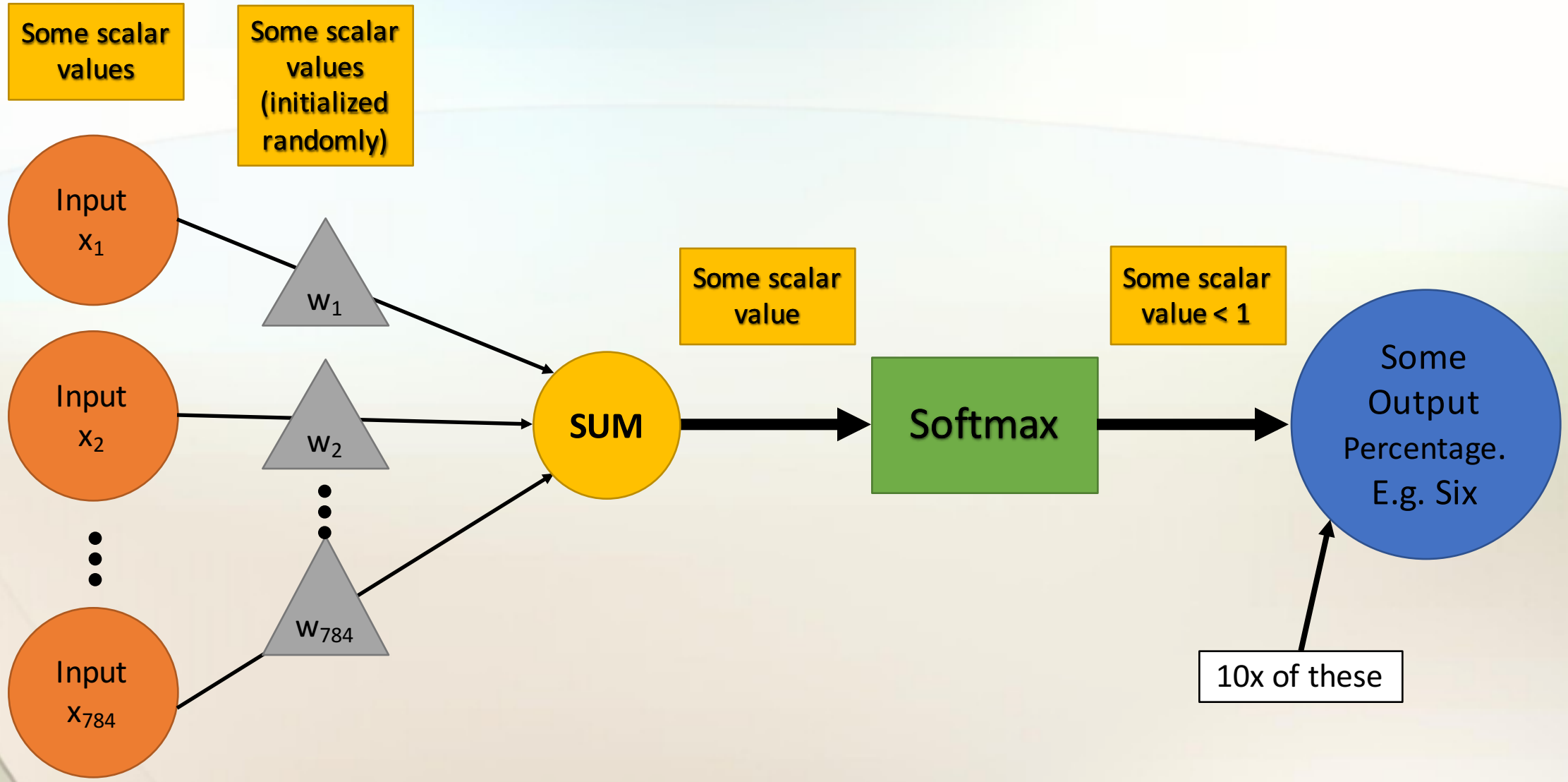
Source: https://ml4a.github.io/ml4a/looking_inside_neural_nets/

Define Our Model

- Each pixel will have 10 associated weights (1 for each layer)
- There are 784 pixels in each image
- Our weight matrix will have dimensions 784 by 10



Artificial Neuron (Simplified)

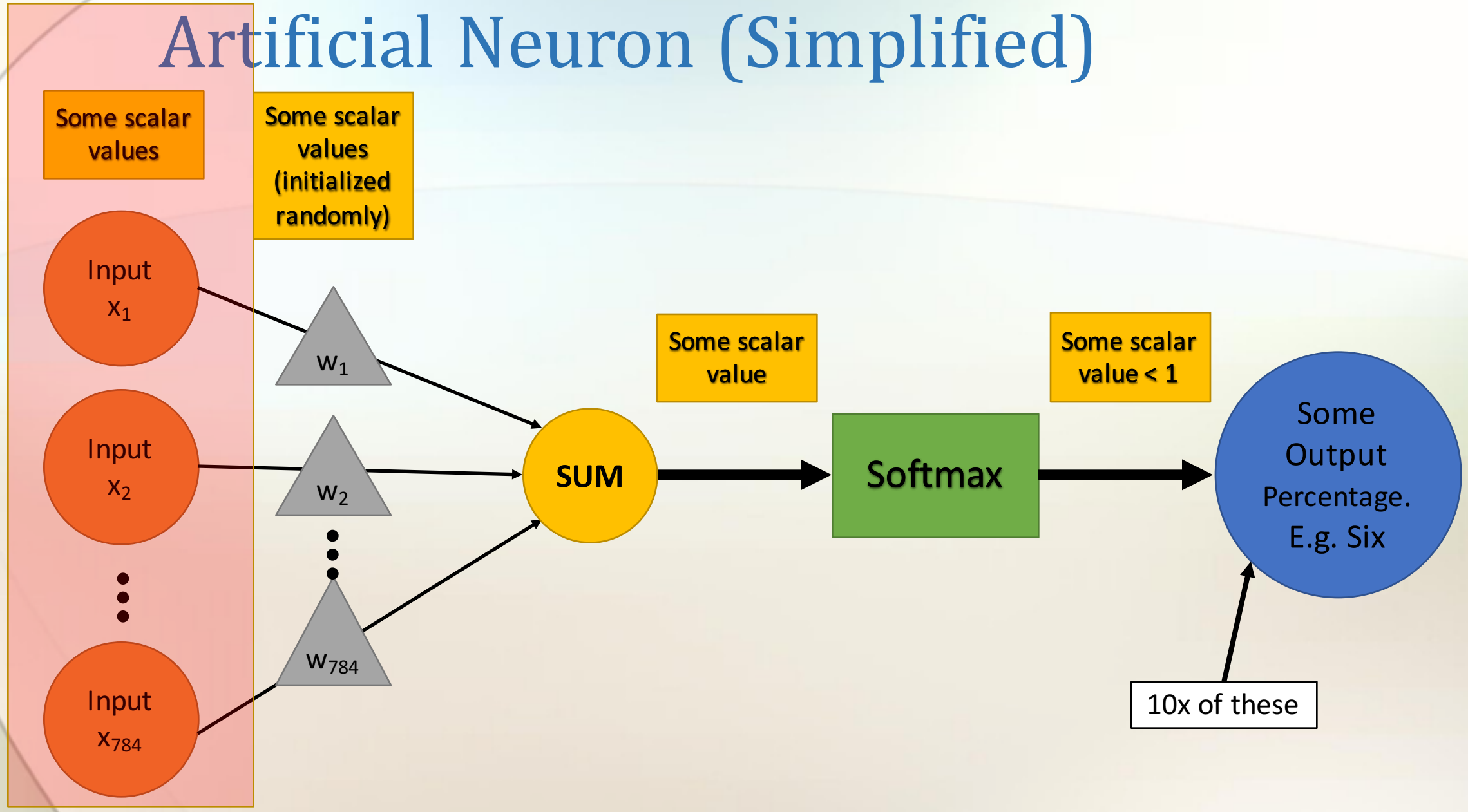


Input Variable

```
x = tf.placeholder(tf.float32, [None, 784])
```

- Creates a placeholder variable “x”
  - “x” doesn’t have a specific value yet
  - Its just a variable, like in math
- Placeholder for our input images
- It is of type “TensorFlow Float 32”
- It has shape “None” by 784
  - None means the first dimension can have any length
  - 784 is the size of one image

Artificial Neuron (Simplified)

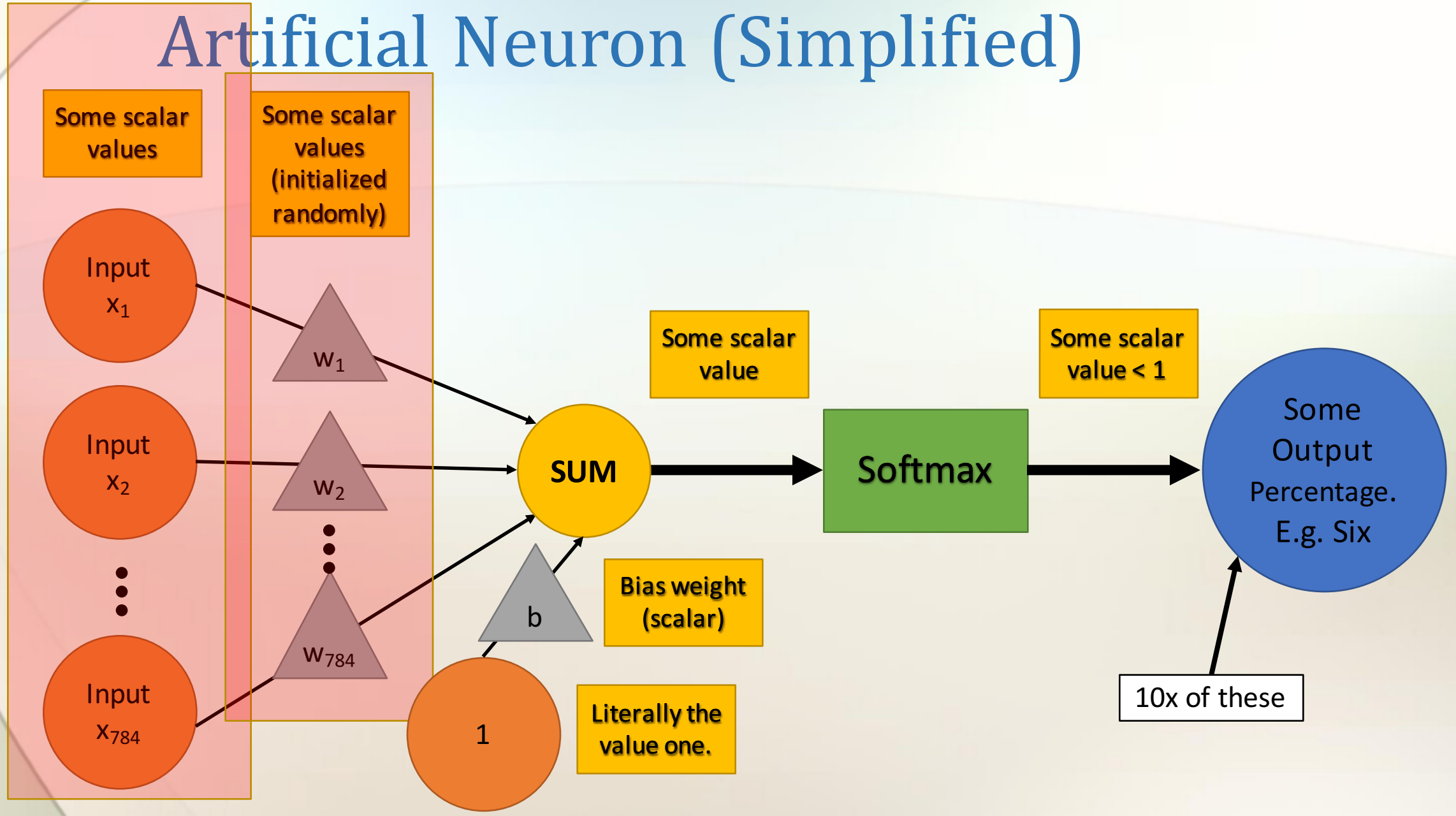


Network Variables (Weights)

```
W = tf.Variable(tf.zeros([784,10]))
```

- Creates a variable W (for “weight”) of size 784 by 10
  - All elements of W are set to 0
  - Unlike “placeholder”, Variable contains determined values

Artificial Neuron (Simplified)

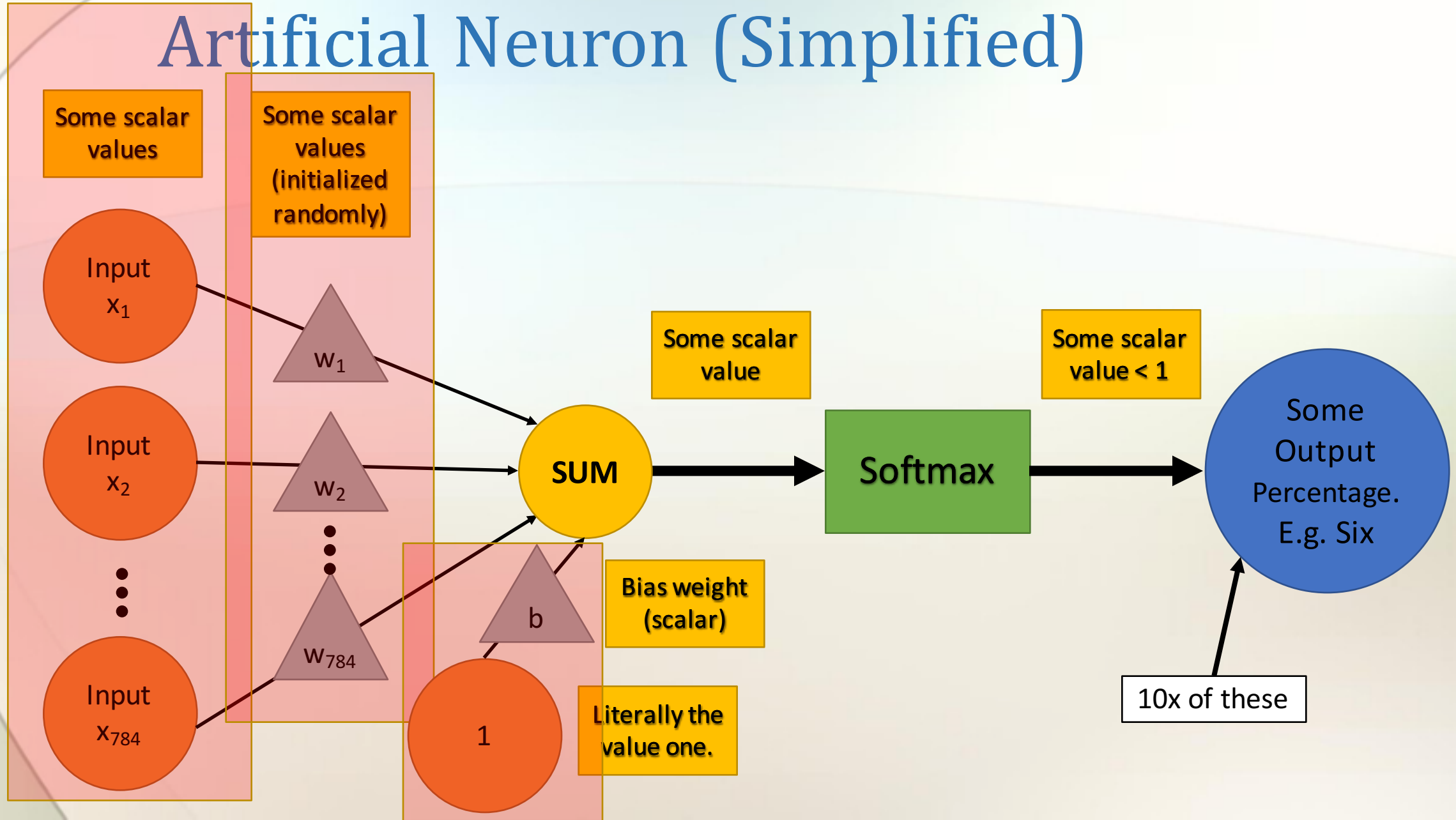


Network Variables (Biases)

```
b = tf.Variable(tf.zeros([10]))
```

- Creates a variable b (for “bias”) of size 10 (by 1)
  - All elements of b are set to 0
  - Unlike “placeholder”, Variable contains determined values

Artificial Neuron (Simplified)

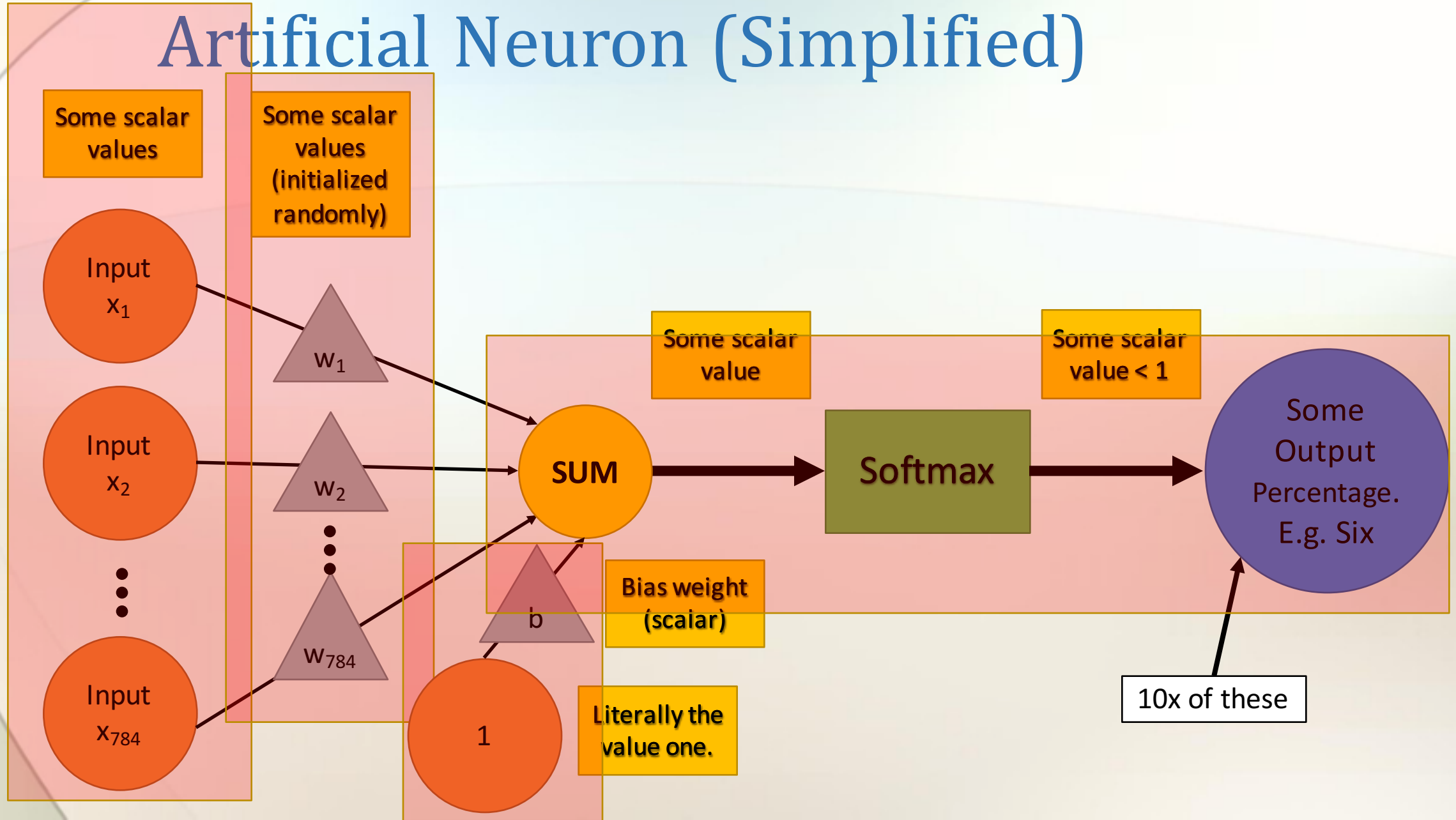


Network Output Variables

```
y = tf.nn.softmax(tf.matmul(x, W) + b)
```

- `tf.matmul(x, W)` performs a matrix multiplication between input variable “x” and weight variable W
- `tf.matmul(x, W) + b` add the bias variable
- `tf.nn.softmax(tf.matmul(x, W) + b)` perform the softmax operation
- `y` will have dimension None by 10

Artificial Neuron (Simplified)



Ground Truth Output Variables

```
yTruth = tf.placeholder(tf.float32, [None, 10])
```

- Creates a placeholder variable “yTruth”
  - “y” doesn’t have a specific value yet
  - Its just a variable, like in math
- Placeholder for Ground Truth one-hot label outputs
- It is of type “TensorFlow Float 32”
- It has shape “None” by 10
  - None means the first dimension can have any length
  - 10 is the number of classes

Loss Variable

```
loss = tf.reduce_mean(-tf.reduce_sum(yTruth * tf.log(y),
                                     reduction_indices=1))
```

- `tf.log(y)` turns values close to 1 to be close to 0, and values close to 0 to be close to –infinity
- `yTruth*tf.log(y)` only keeps the value of the actual class
- `-tf.reduce_sum(yTruth*tf.log(y),reduction_indices=1))` Sums along the class dimension (mostly 0's), fixes the sign
- `tf.reduce_mean( … )` averages the vector into a scalar

Loss Variable Example

Predict (y)	
Cat	Dog
0.25	0.75
0.90	0.10
0.6	0.4

Average
Loss
0.4363

Ground Truth (yTruth)	
Cat	Dog
0	1
1	0
0	1

Implement Training Algorithm

```
lr = 0.5 # learning rate
trainStep = tf.train.GradientDescentOptimizer(lr).minimize(loss)
```

- Learning rate is how much to proportionally change weights per training example.
- Minimize the loss function
- *Magic*

Begin the TensorFlow Session

- Up to this point, we have just been laying down a blueprint for TensorFlow to follow, but it hasn't "built" anything yet.

```
init = tf.global_variables_initializer()
sess = tf.Session()
sess.run(init)
```

- Initialize variables
- Create and run a TensorFlow session.

Run Through the Training Dataset

```
batchSize = 100
for i in range(1000):
    # get some images and their labels
    xBatches, yBatches = mnist.train.next_batch(batchSize)
    sess.run(trainStep,feed_dict={x:xBatches,yTruth:yBatches})
```

- Repeat 1000 times
- Gets a small random (100) subset of our training dataset
- Train on that small subset (this line updates the weights)
- Hopefully have a trained network once its done looping!

How Well Does It Perform?

```
correctPred = tf.equal(tf.argmax(y,1), tf.argmax(yTruth,1))
accuracy = tf.reduce_mean(tf.cast(correctPred, tf.float32))
resultAcc = sess.run(accuracy, feed_dict=
            {x: mnist.test.images, yTruth: mnist.test.labels})
print("Trained Acc: %f" % resultAcc)
```

- Approximately 92% accurate
- YOU'VE DONE IT! YOU'VE DONE DEEP LEARNING!!!
  - Kind of, this was a super small, simple, shallow network.
    - 92% is quite bad on this problem
    - Best systems are around 99.7% accurate (Convolutional Neural Networks).

Going Further

- Two main areas to work on machine learning:
  - Architecture, the better the architecture the better the results
    - More layers
    - “Fatter” layers
    - Intertwining layers
    - Tricks like dropout, dropconnect, regularization, pooling, maxout, etc.
  - Network Building Blocks
    - Convolutional Neural Networks
    - Recurrent Neural Networks
    - Generative Adversarial Neural Networks

Questions?