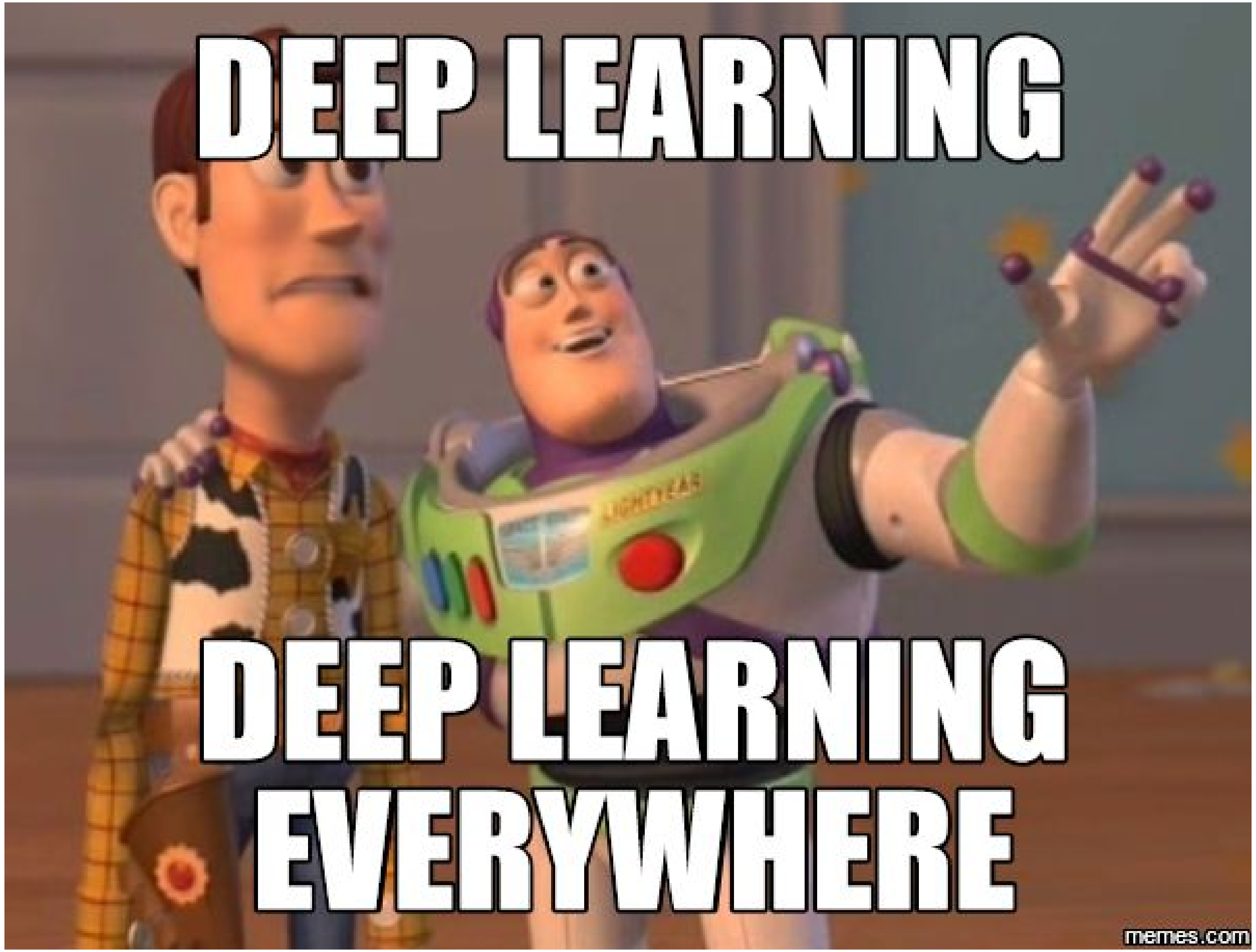
GENERAL ASSEMBLY

Intro to Artificial Neural Networks

Patrick D. Smith

Artificial Neural Networks



LEARNING OBJECTIVES

- Understand the different types and uses of Artificial Neural Networks
- Learn and understand the structure of artificial neural networks
- Learn how to manually construct an artificial networks
- Implement a neural network utilizing the Keras library

OPENING

Artificial Neural Networks

Artificial Neural Networks

What is deep learning?

- In a nutshell, deep learning is the practicing of creating artificial neural networks that are composed of many layers

Artificial Neural Networks

What is deep learning?

- In a nutshell, deep learning is the practicing of creating artificial neural networks that are composed of many layers
- We can think of deep learning is the intersection of optimization and functional programming

Artificial Neural Networks

What is deep learning?

- In a nutshell, deep learning is the practicing of creating artificial neural networks that are composed of many layers
- We can think of deep learning is the intersection of optimization and functional programming
- An algorithm must pass data through several non-linear states to be considered a “deep learning” algorithm.

Artificial Neural Networks

What is deep learning?

- In a nutshell, deep learning is the practicing of creating artificial neural networks that are composed of many layers
- We can think of deep learning is the intersection of optimization and functional programming
- An algorithm must pass data through several non-linear states to be considered a “deep learning” algorithm.
- Trees, SVMs, and Naive Bayes are all considered shallow algorithms - these are not deep learning techniques

Artificial Neural Networks

What is deep learning?

- In a nutshell, deep learning is the practicing of creating artificial neural networks that are composed of many layers
- We can think of deep learning is the intersection of optimization and functional programming
- An algorithm must pass data through several non-linear states to be considered a “deep learning” algorithm.
- Trees, SVMs, and Naive Bayes are all considered shallow algorithms - these are not deep learning techniques
- The goal of all of this? **To bring us one step closer to true AI**

Artificial Neural Networks

Deep Learning Has Been Around For Some Time Now

- The foundations for deep learning have been around ***since 1943***
- First advancements were with the **perceptron** in the 1950s
- Artificial Neural Networks started evolving in the 1980s (CNNs & Yann LeCun)
- LSTM, arguably the most researched and powerful network structure at the moment, has been around ***since 1997***

Artificial Neural Networks

Why use deep learning?

1. Deep learning almost completely eliminates the need for feature engineering and selection. As data is passed through the network, it becomes more abstracted and dissected for us.

Artificial Neural Networks

Why use deep learning?

1. Deep learning almost completely eliminates the need for feature engineering and selection. As data is passed through the network, it becomes more abstracted and dissected for us.
2. **ANNS PERFORM BETTER THAN ALMOST EVERY OTHER MODEL**

Artificial Neural Networks

Tasks where ANNs are scientifically proven to perform better:

- **Handwriting recognition**
- **Speech recognition**
- **Traffic Analysis**
- **Anomaly detection for computer vision**
- **Human action recognition**
- **Detecting Cancer and other other diseases**
- **The entire backend of your facebook feed**
- **Customer segmentation**
- **Life**

Artificial Neural Networks

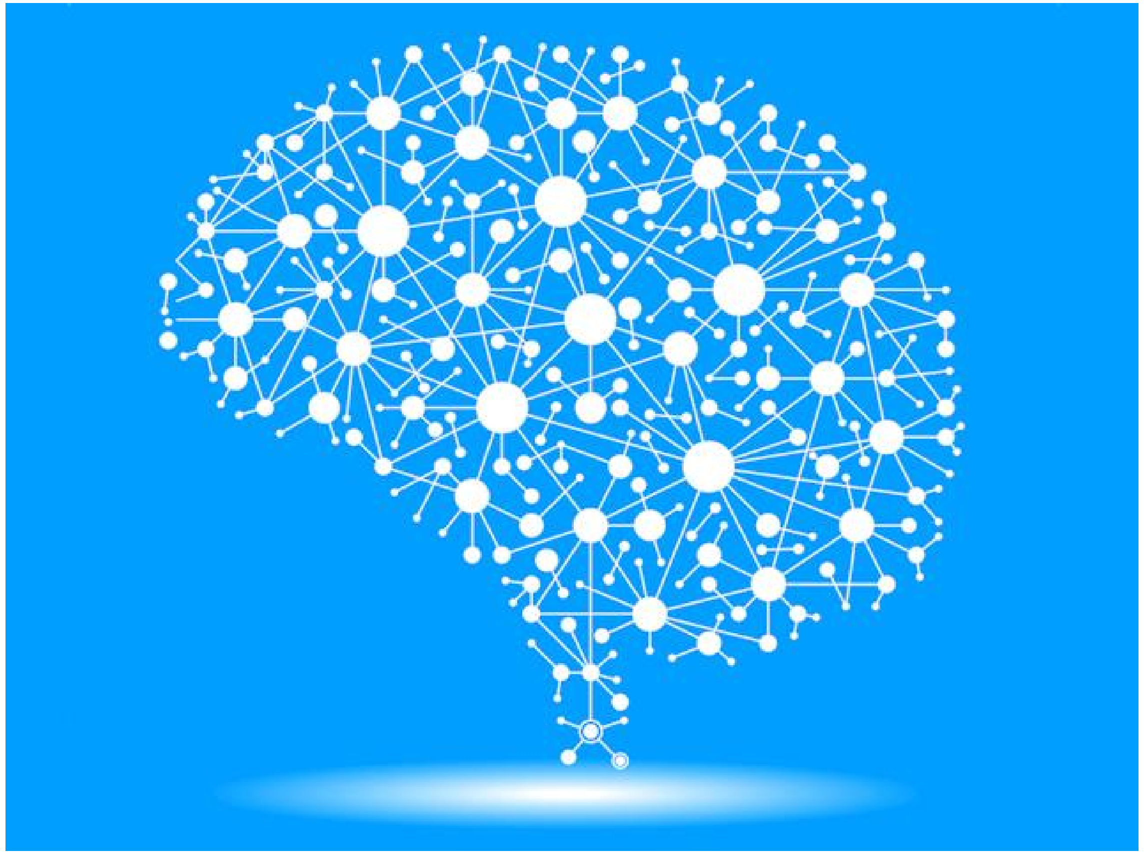
When creating an ANN; there are three approaches one can take:

- **The biological approach:** These networks draw their structure directly from biology and the structure of the brain's frontal cortex.
- **The representative approach**: This approach looks at actual data transformations from the perspective of the **manifold hypothesis**
- **The probabilistic approach:** This approach sees neural networks as finding latent variables (variables that are not directly oberserved, but rather infered)

Artificial Neural Networks

We're going to break this lesson down into three sections:

- **Section I:** A generalized approach to artificial neural networks and their structure
- **Section II:** Convolutional Neural Networks
- **Section III:** Recurrent Neural Networks

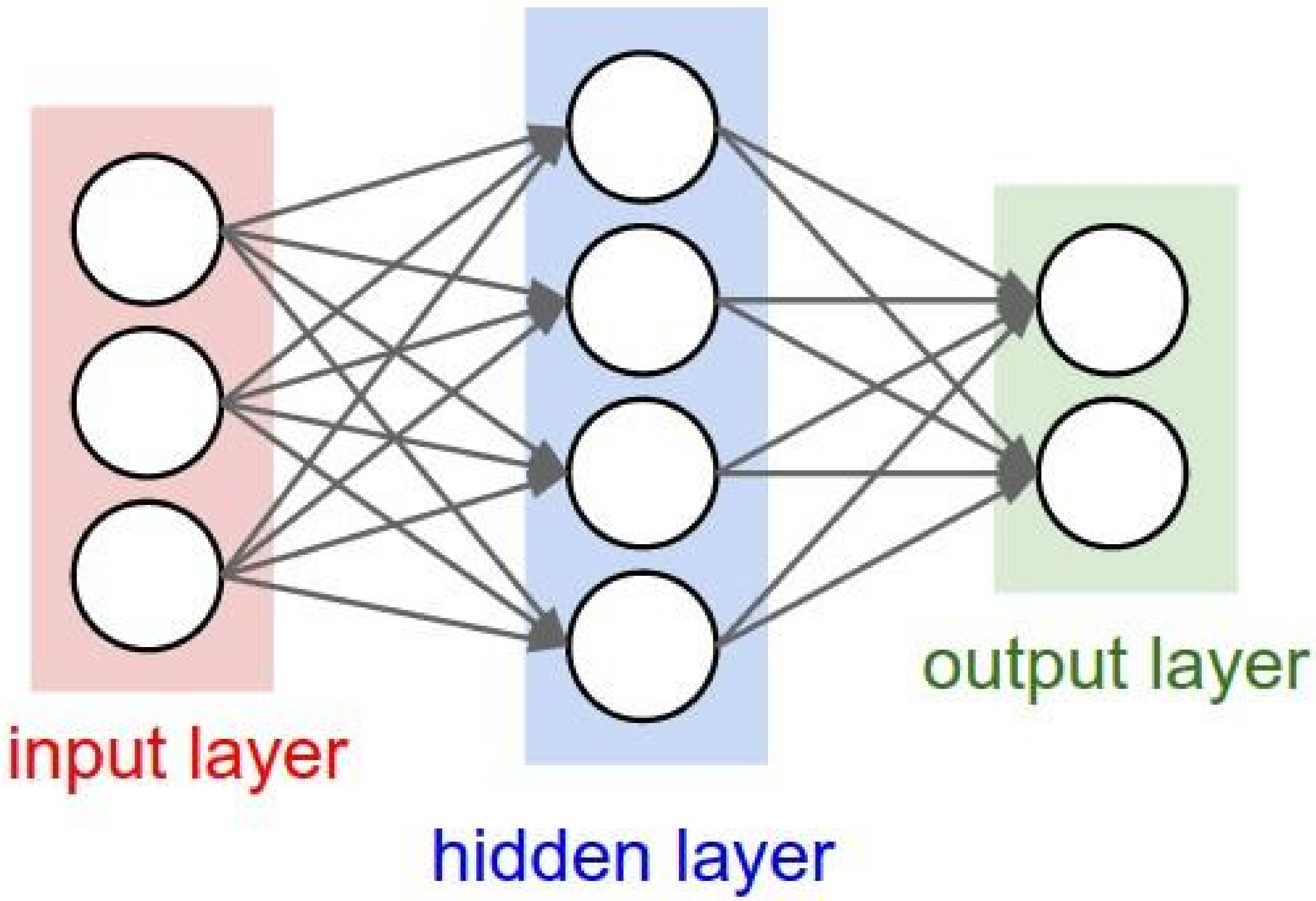


Part I: A generalized approach to ANNs

Artificial Neural Networks: General Structure

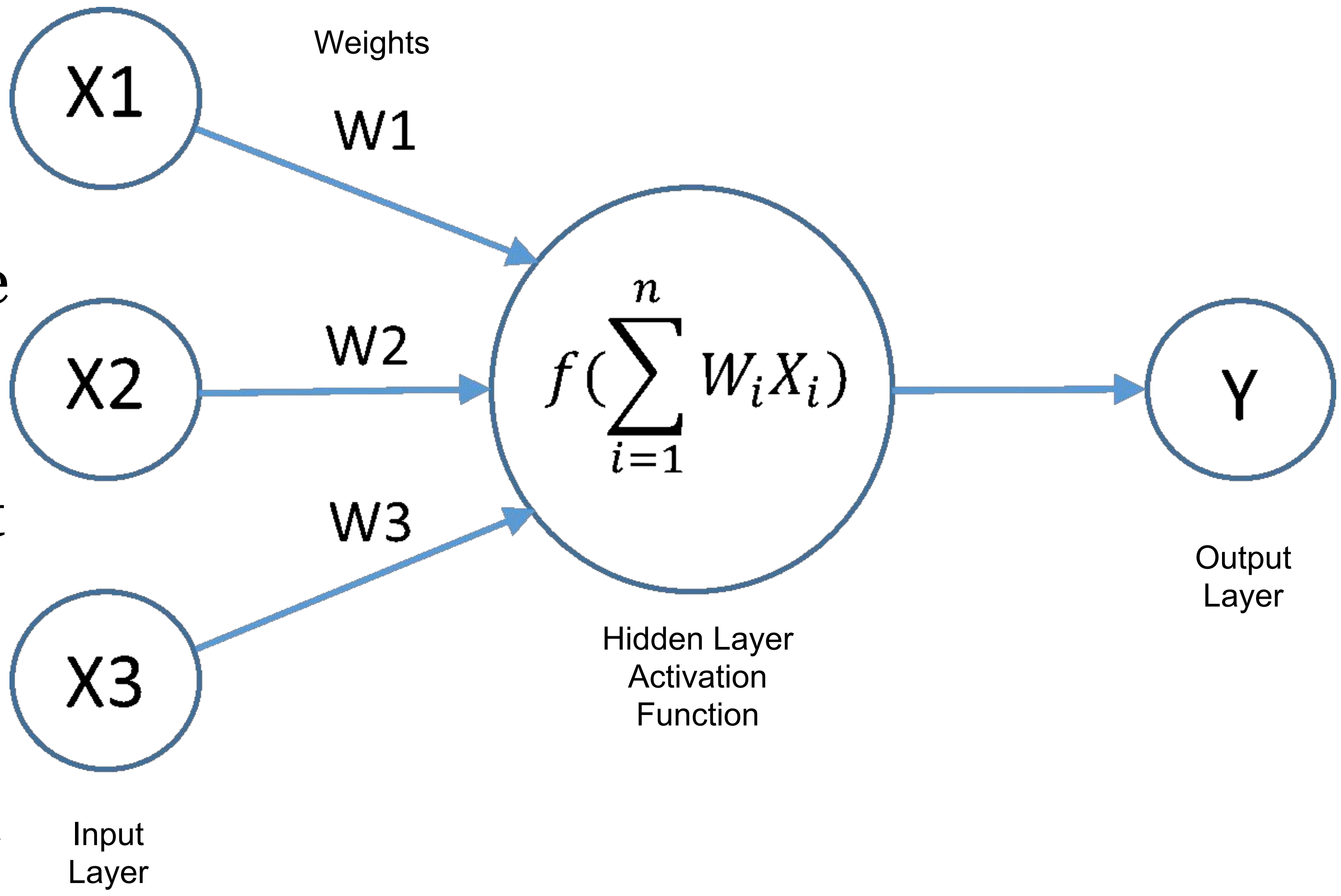
- ANN's have a a basic structure of three elements:
- **The Input Layer**
- **The Hidden Layer**
- **The Output Layer**

- The input layer is a ***passive layer***, while the hidden and output layers are ***active layers***



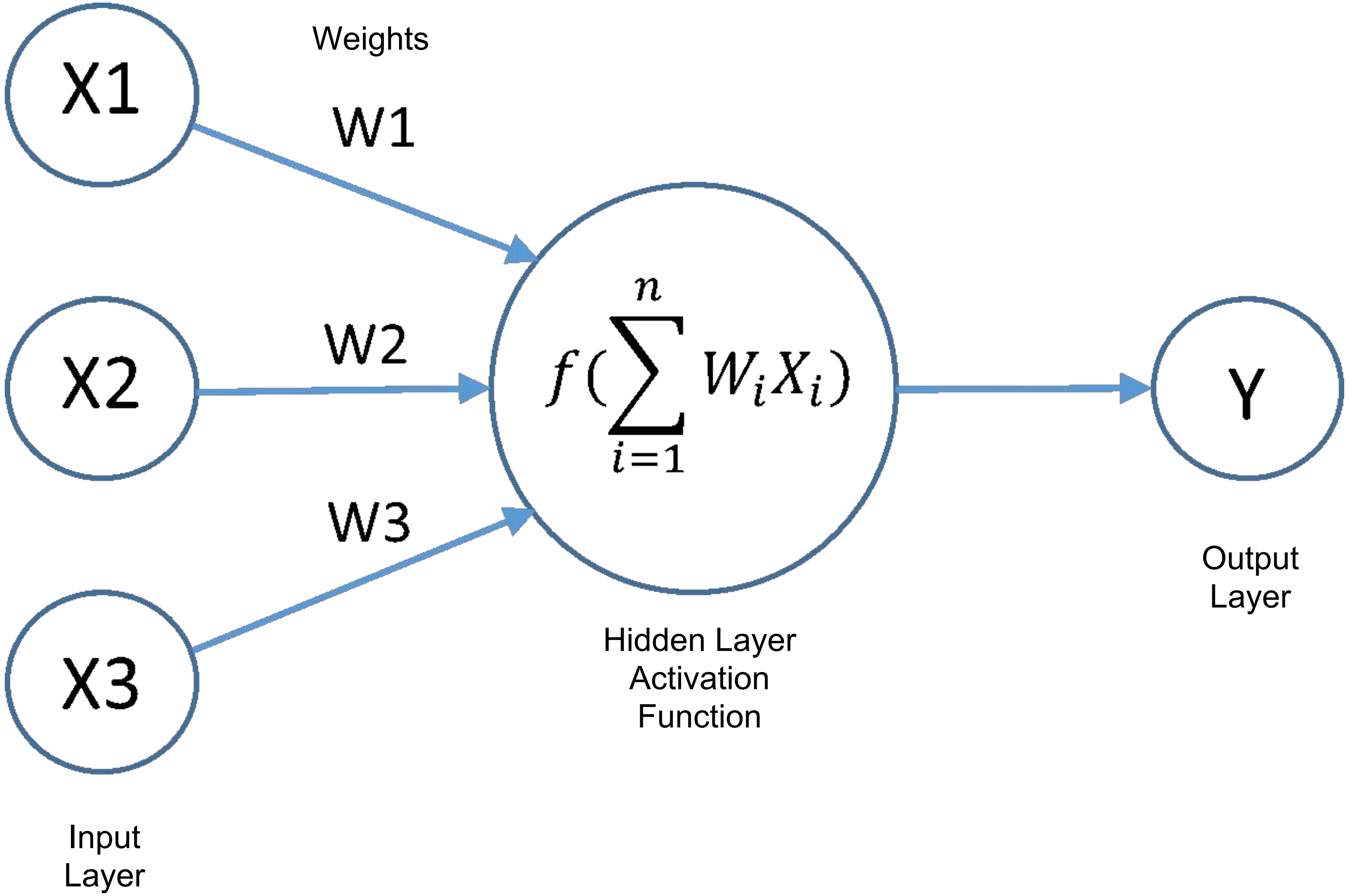
Artificial Neural Networks: General Structure

- The lowest layer takes the raw data like images, text, sound, etc. and then each neurons stores some information about the data they encounter.
- As data moves to the next layers, neurons learn a more abstract version of the data below it.
- The higher you go up, the more abstract features you learn



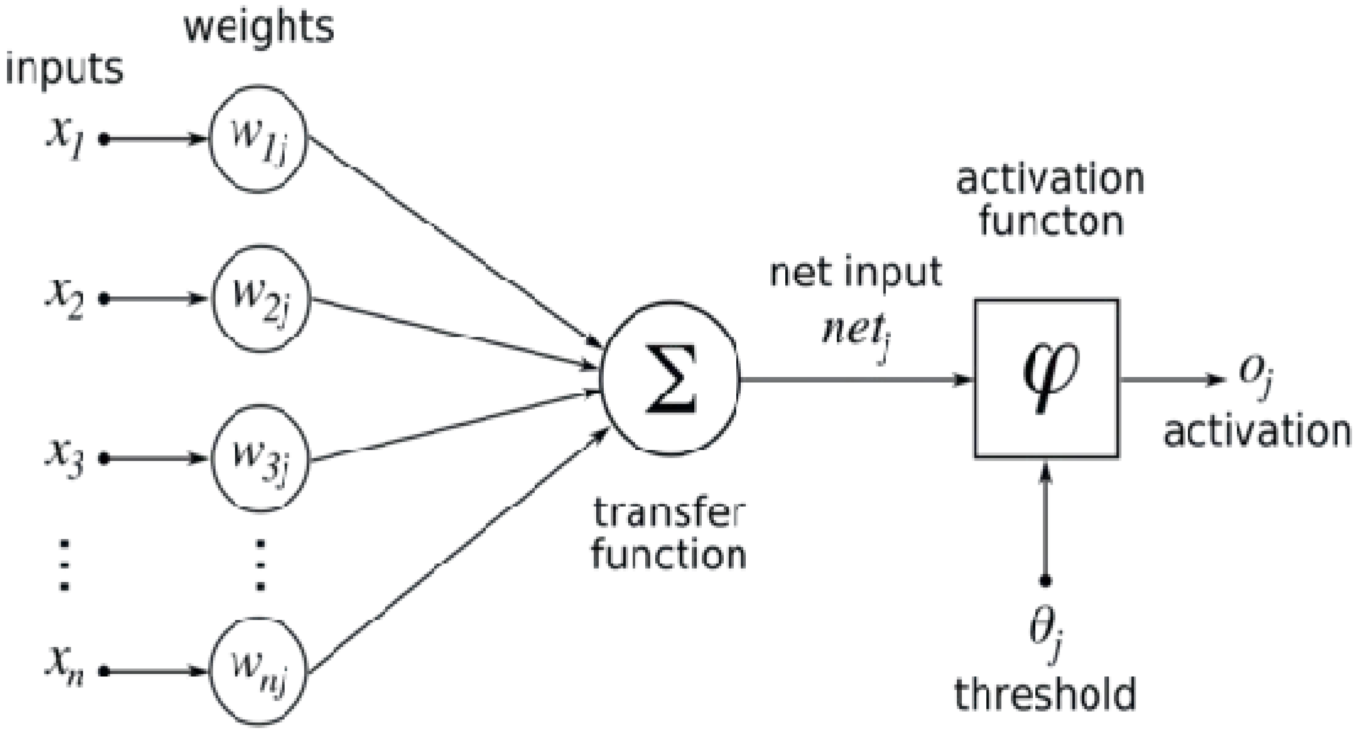
Artificial Neural Networks: General Structure

- Each value from the input layer is duplicated and sent to the hidden layer, resulting in a fully connected network
- The values from the input layer are multiplied by ***weights*** before they are sent to the hidden layer



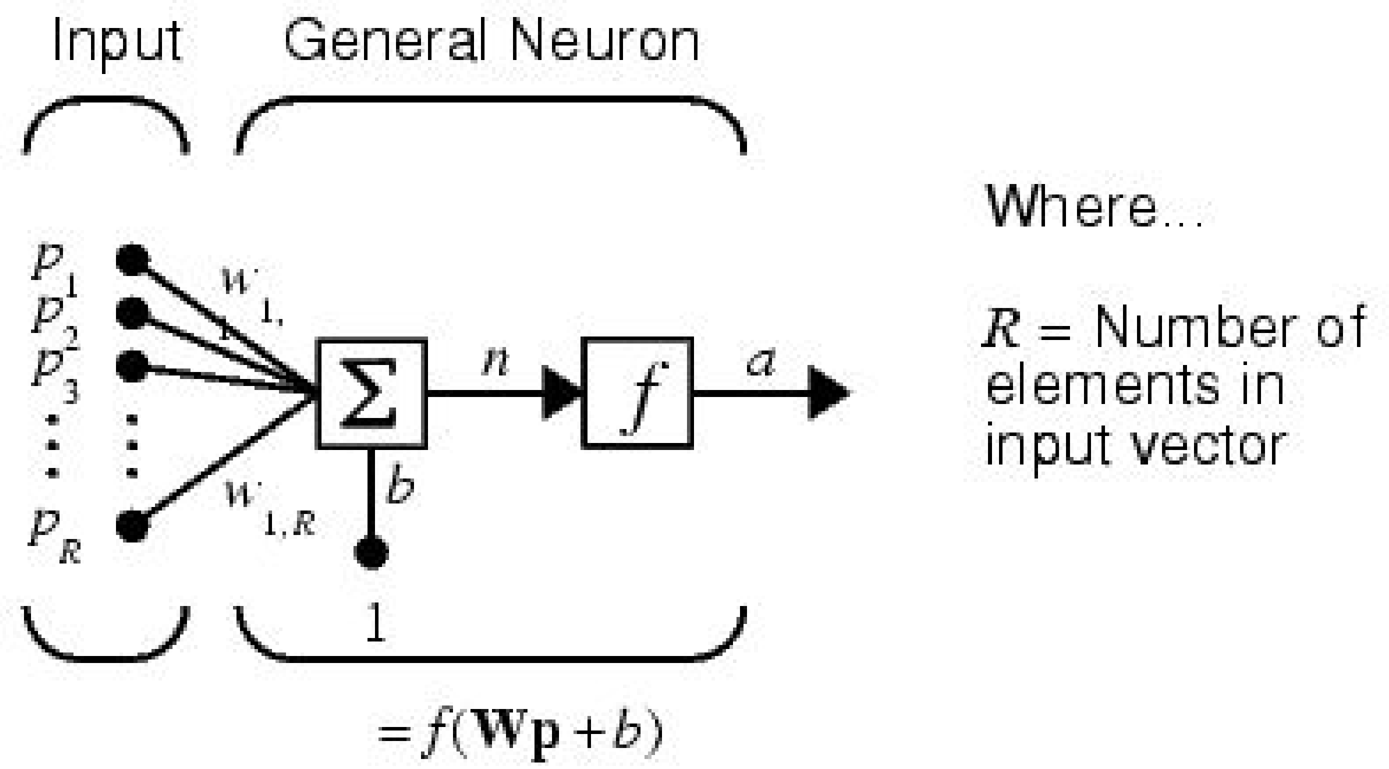
Artificial Neural Networks: General Structure

- These weights are summed in a **transfer function** and then sent to an **activation function**
- The activation function essentially answers a question: "*Shall we turn on the output switch?*"



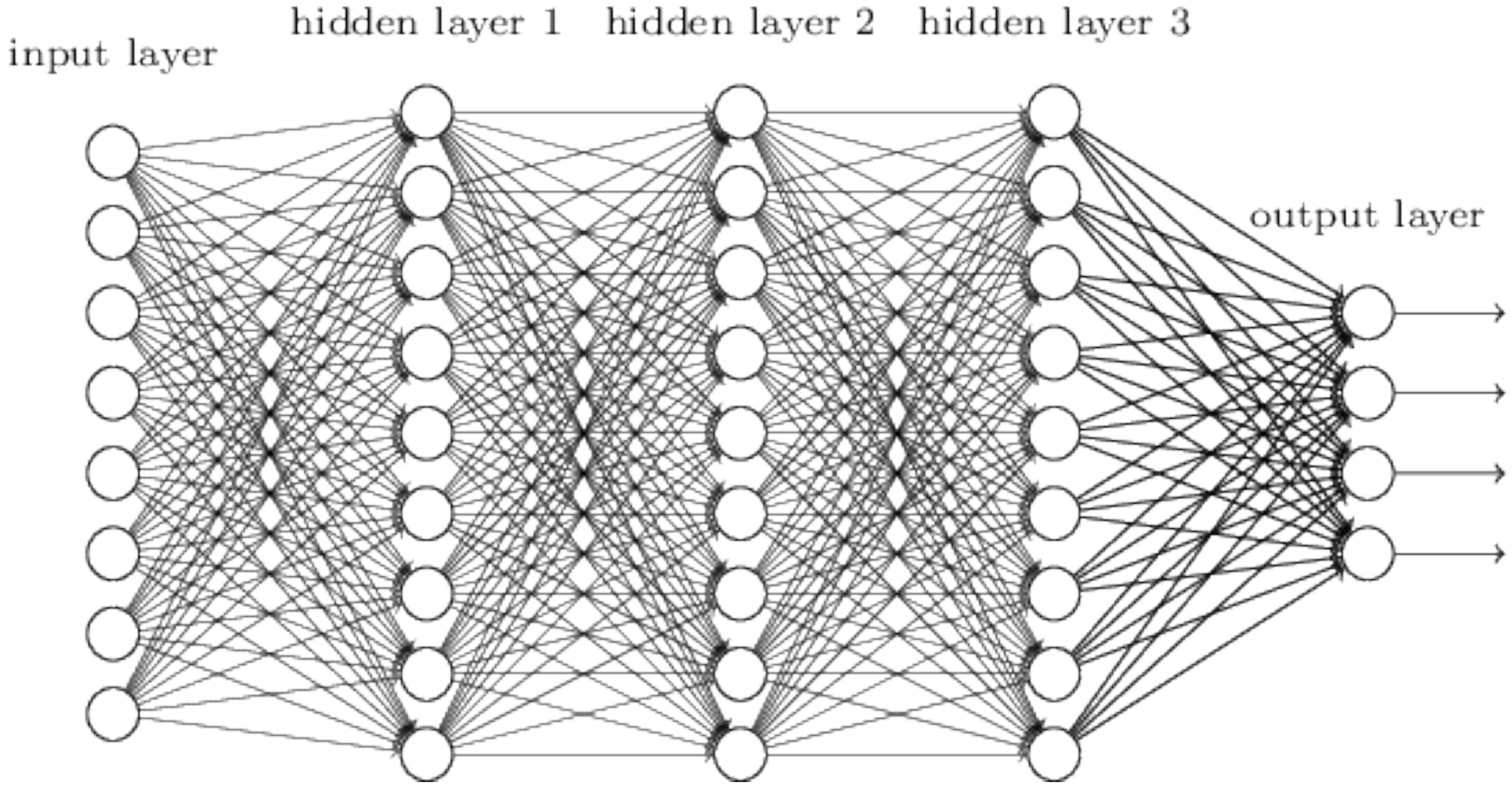
Artificial Neural Networks: General Structure

- Alternatively, we can pass the input to a non-linear transformation function, such as a **sigmoid function** or **softmax function** (denoted here by function ***f)*** in place of an activation function



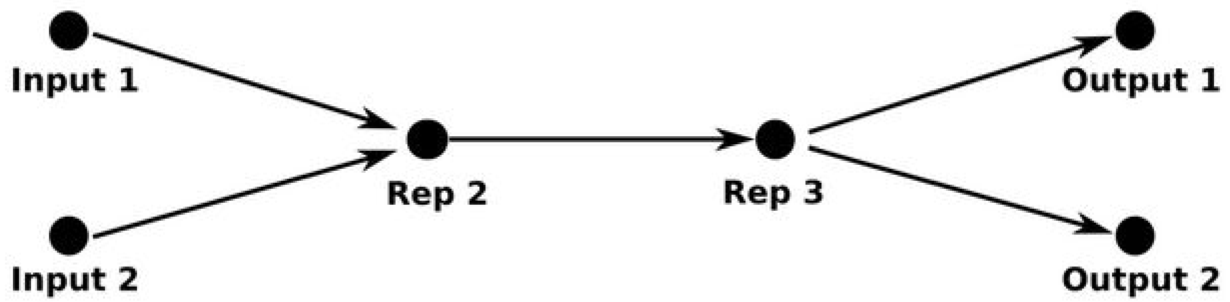
Artificial Neural Networks: General Structure

- The activation function then passes the data anto the hidden layers
- It's important to note that ANNs can have *ANY* number of hidden layers with *ANY* number of nodes



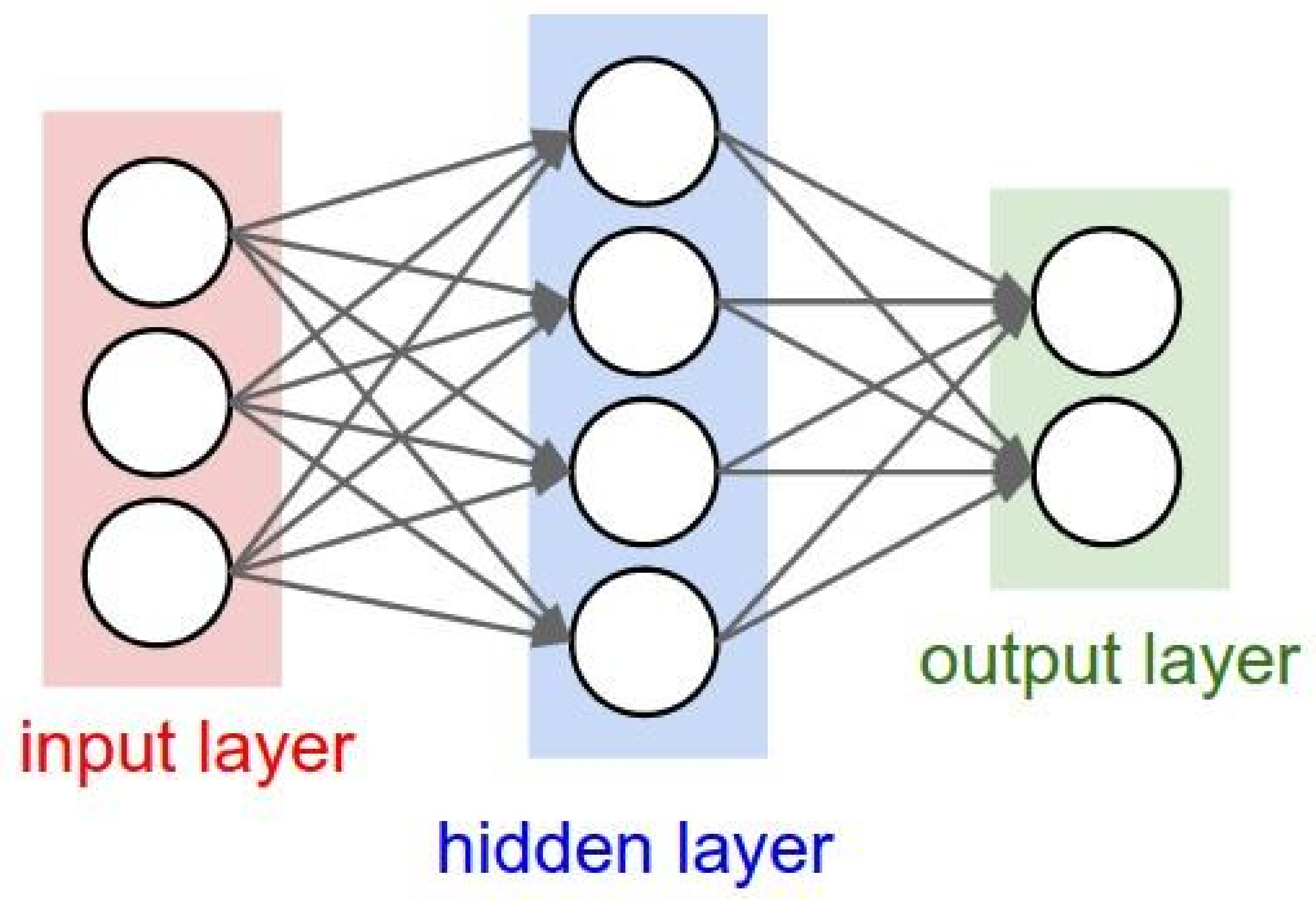
Artificial Neural Networks: General Structure

- Each layer of the networks transforms the data in some manner



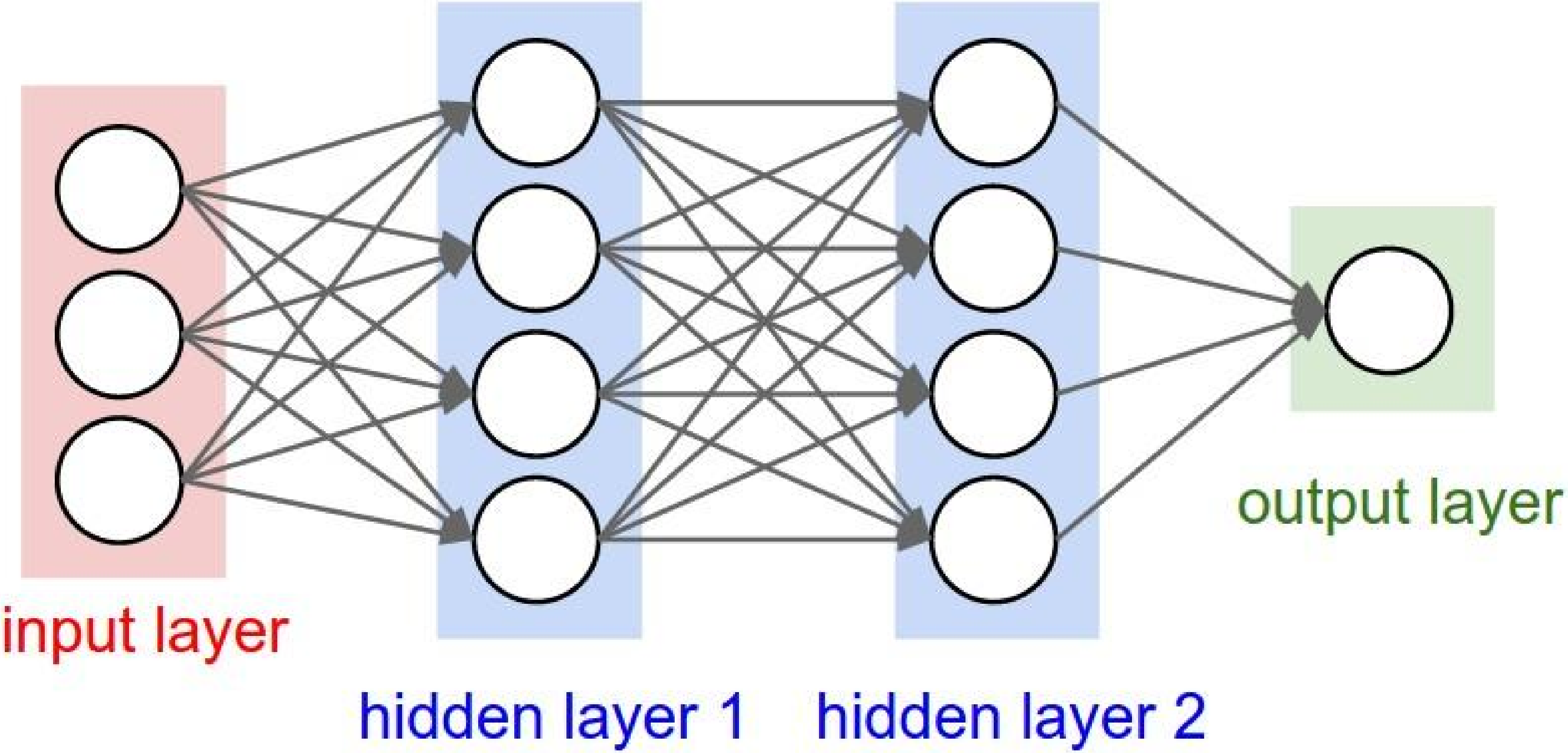
Artificial Neural Networks: General Structure

- Finally, data is passed to the **output layer**
- These layers do not have activation functions as they are usually taken to represent the class scores (e.g. in classification), which are arbitrary real-valued numbers, or some kind of real-valued target (e.g. in regression).



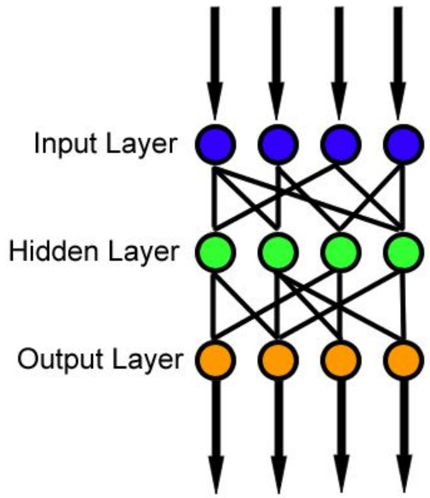
Artificial Neural Networks: General Structure

The error from the first pass is then sent back in the network, and the process repeated.



Artificial Neural Networks: Types of Artificial Neural Networks

We can break neural networks down into two types:

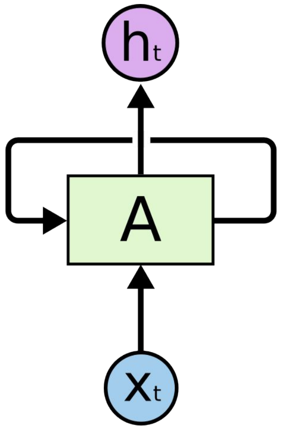


I. Feed Forward Networks

1. Convolutional Neural Networks

II. Recurrent Networks

1. Recurrent Neural Networks
2. Long Short Term Memory Networks
3. Hierarchical Networks



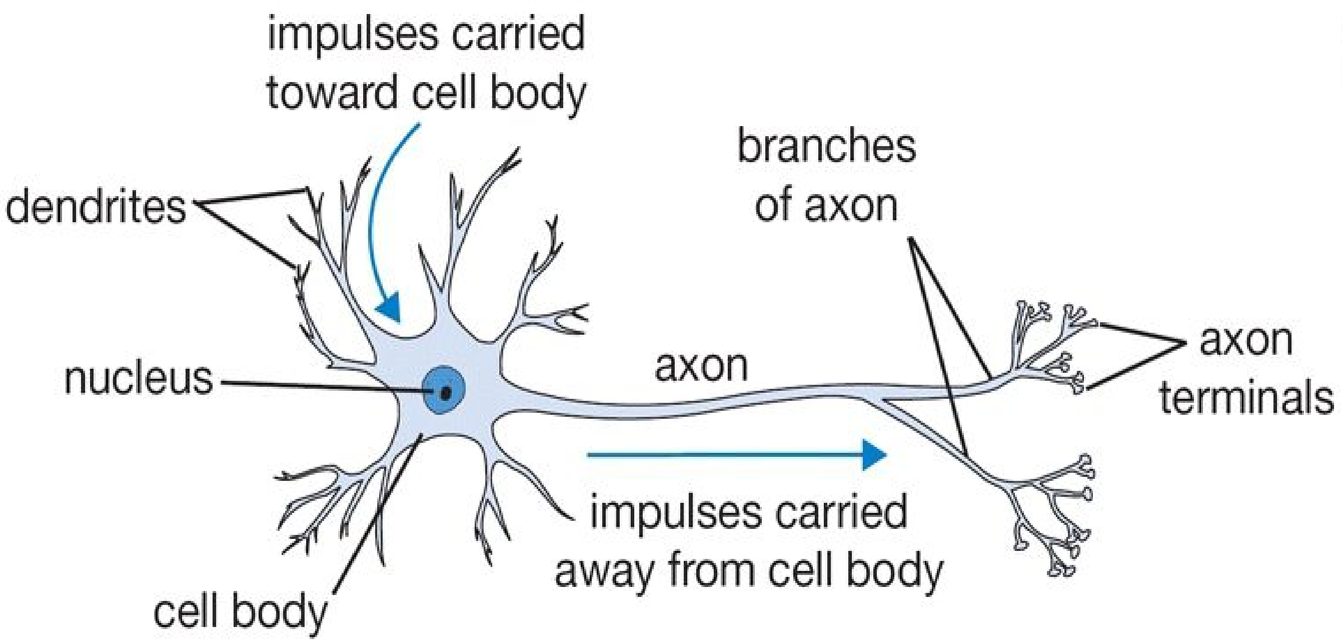
Part II: Basic Elements

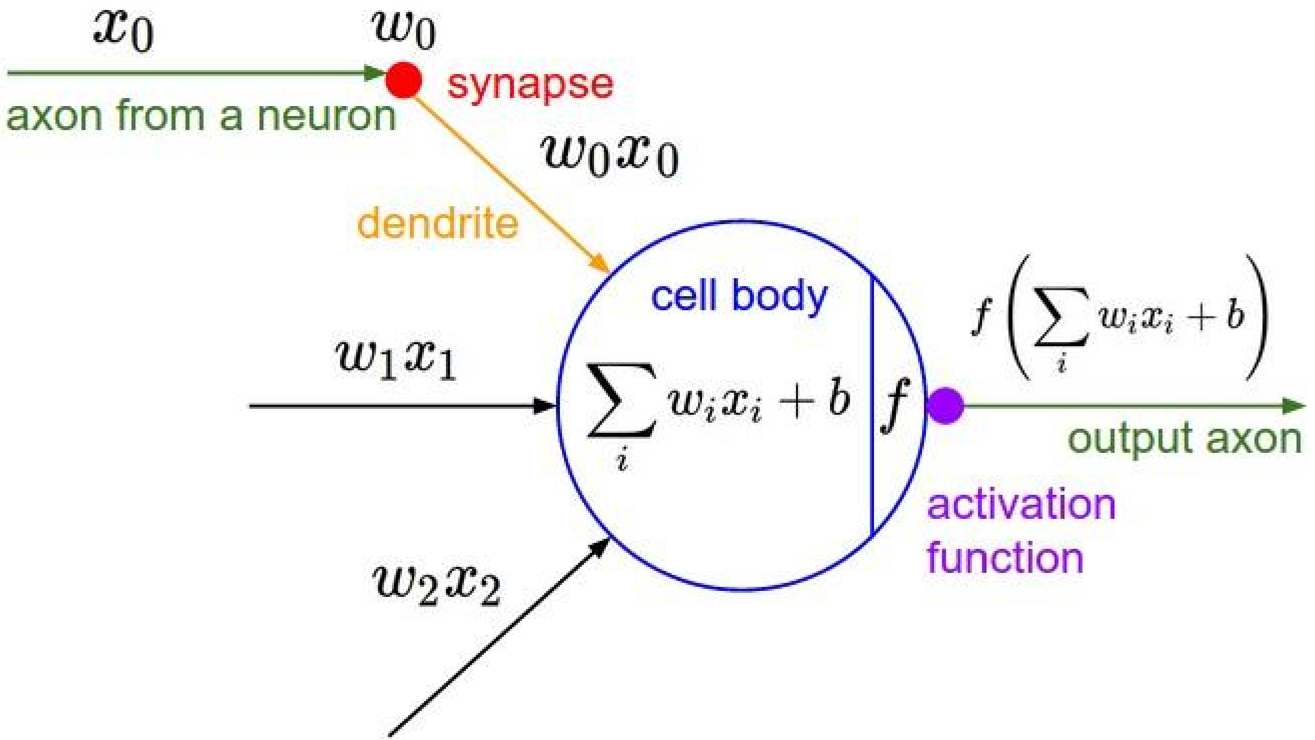
Artificial Neural Networks: Key Elements

Key Elements and Topics in Neural Networks

- **Weights**
- **Biases**
- **Activation Functions**
- **Neurons**
- **Stochastic Gradient Descent**
- **Backpropagation**

Artificial Neural Networks: Key Elements





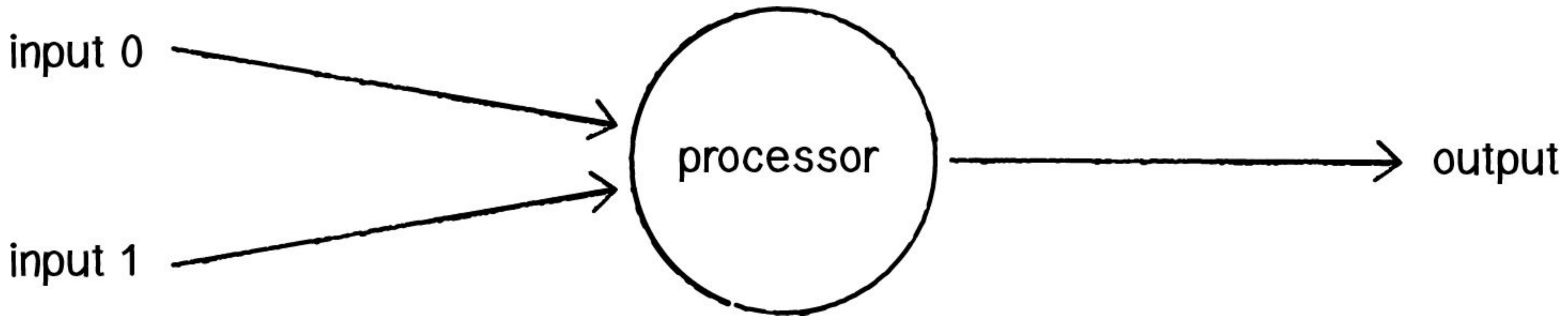
Each neuron performs a dot product with the input and its weights, adds the bias and applies the non-linearity (or activation function)

Weights

Artificial Neural Networks: Types

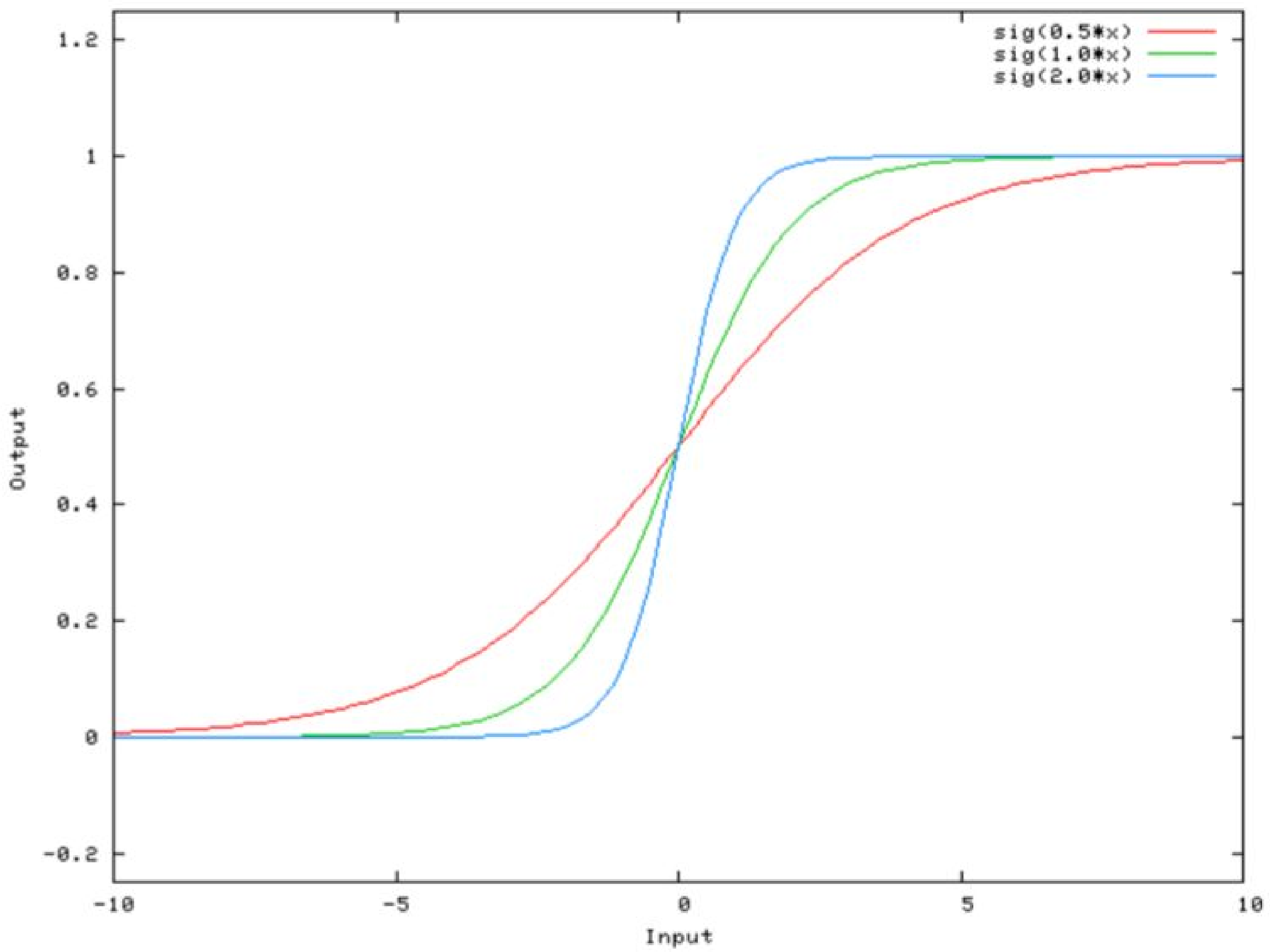
Weights

- Weights in ANNs are values between -1 and 1 that provide some weighting factor to our input
- We typically initialize our weights with an arbitrary value, and these along with the **biases** are dynamically updated by **backpropagation** during the course of learning



Artificial Neural Networks: Types

Weights



Artificial Neural Networks: Types

Weights

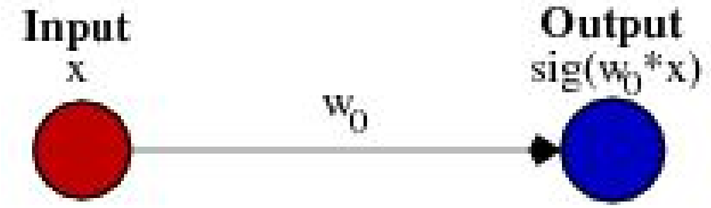
- When we initialize an ANN, we first need to initialize its weights. These will be dynamically updated throughout the training process
- At the end of the training process, we can expect that roughly half our weights will be positive and half of our weights will be negative.
- We don’t want to initialize our weights at 0; because if all of the neurons compute the same output, they might all give the same answer
- It is preferred to initialize the weights as small random numbers

Bias Values

Artificial Neural Networks: Types

Bias Values

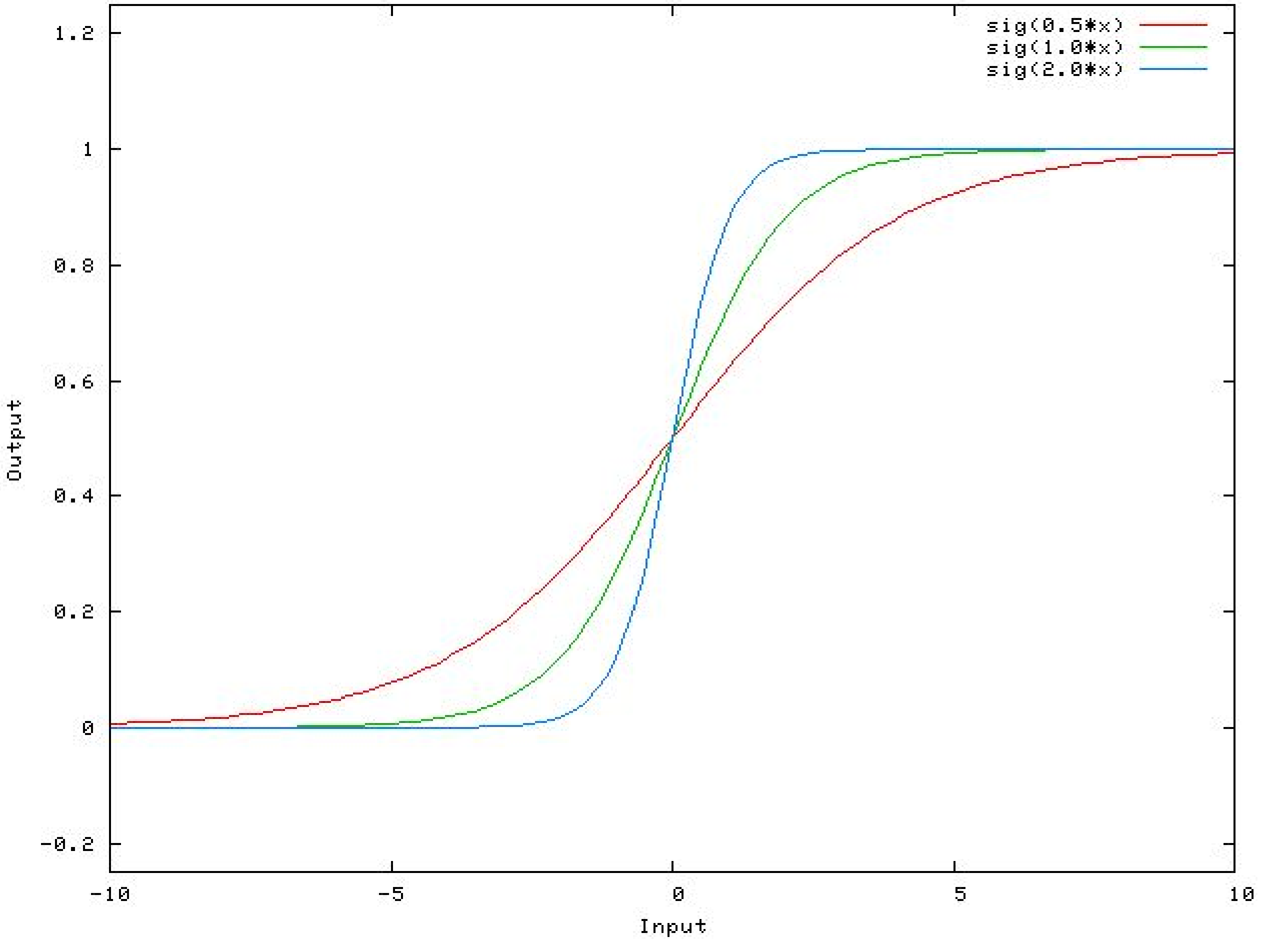
Consider this basic network:



The output of the network is computed by multiplying the input (x) by the weight (w0) and passing the result through the activation function.

Artificial Neural Networks: Types

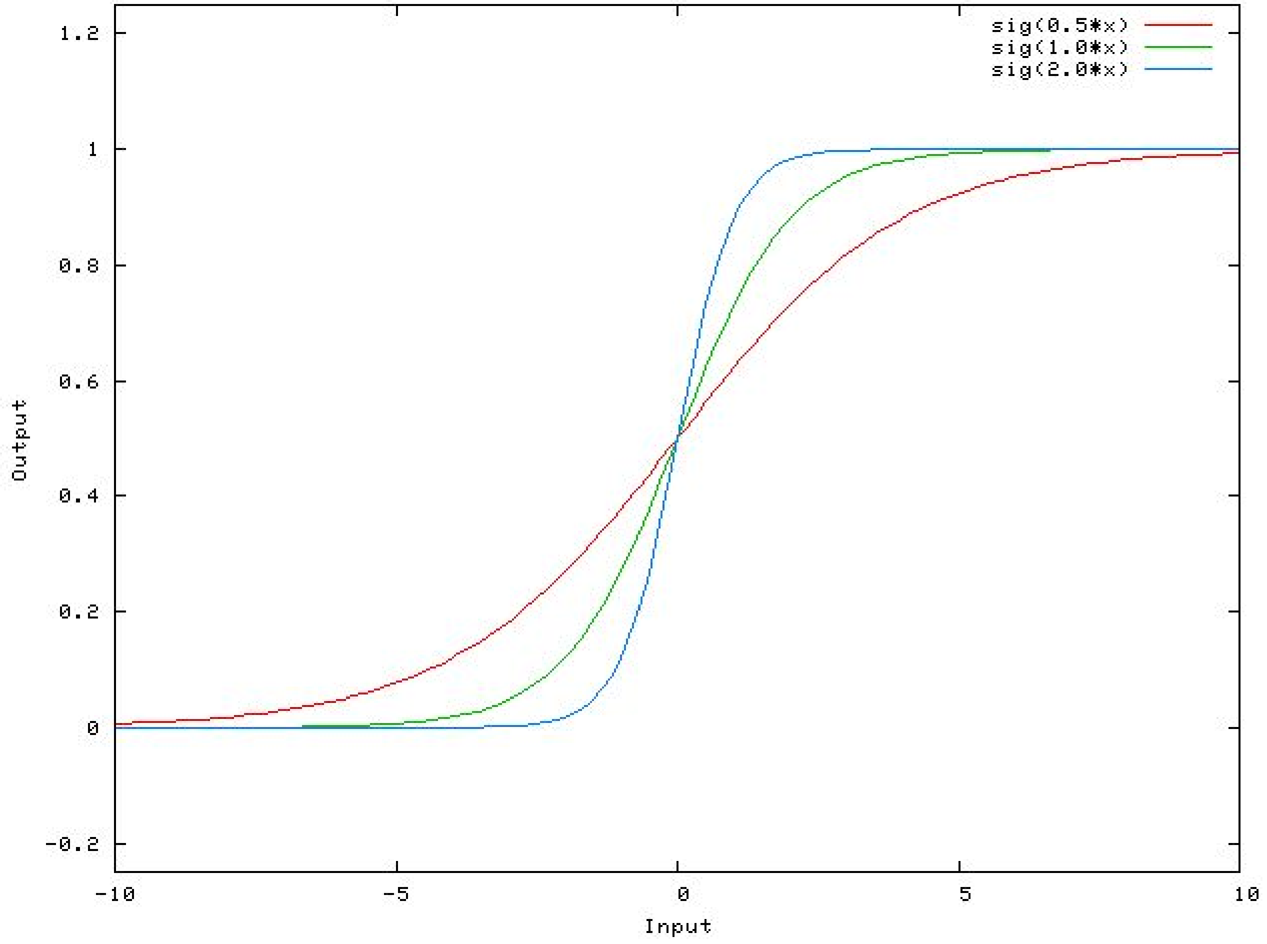
Bias Values



From this network, we should expect a result such as the one on the left

Artificial Neural Networks: Types

Bias Values

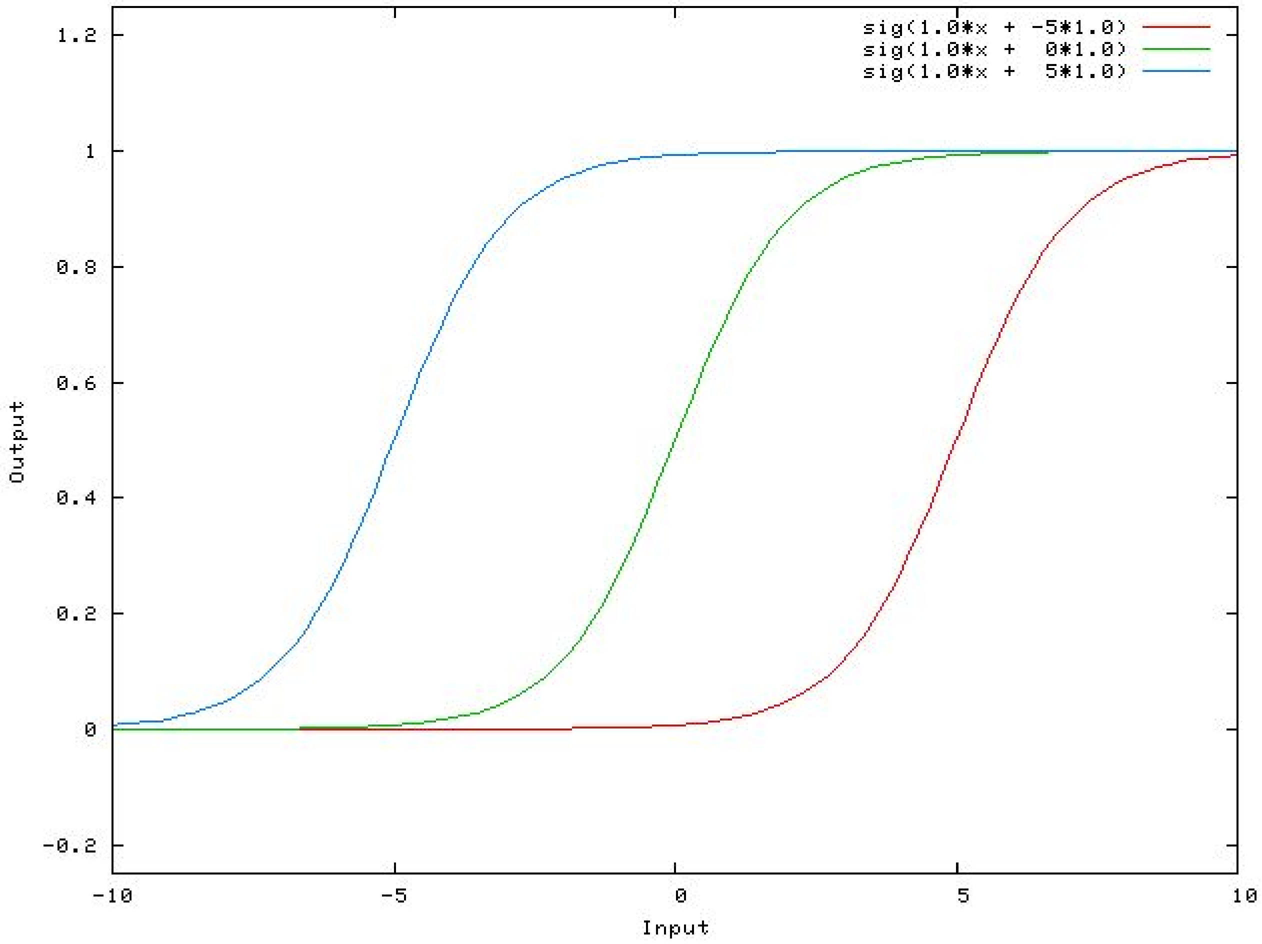


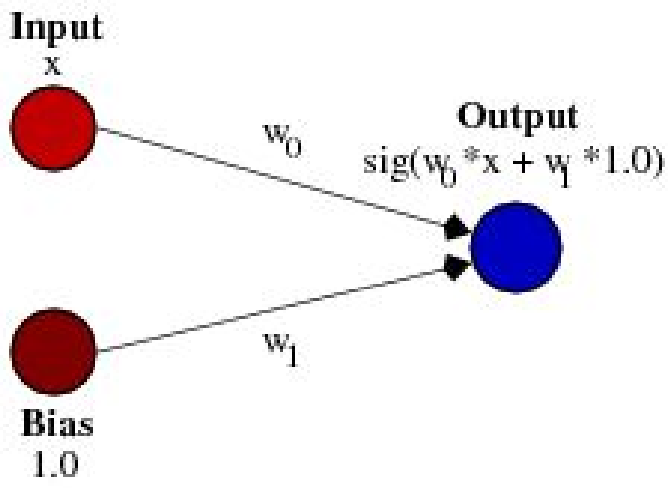
From this network, we should expect a result such as the one on the left

Say, however, that we want a network that will output 0 when the input is 2? We would need to shift this function to the right.

Artificial Neural Networks: Types

Bias Values



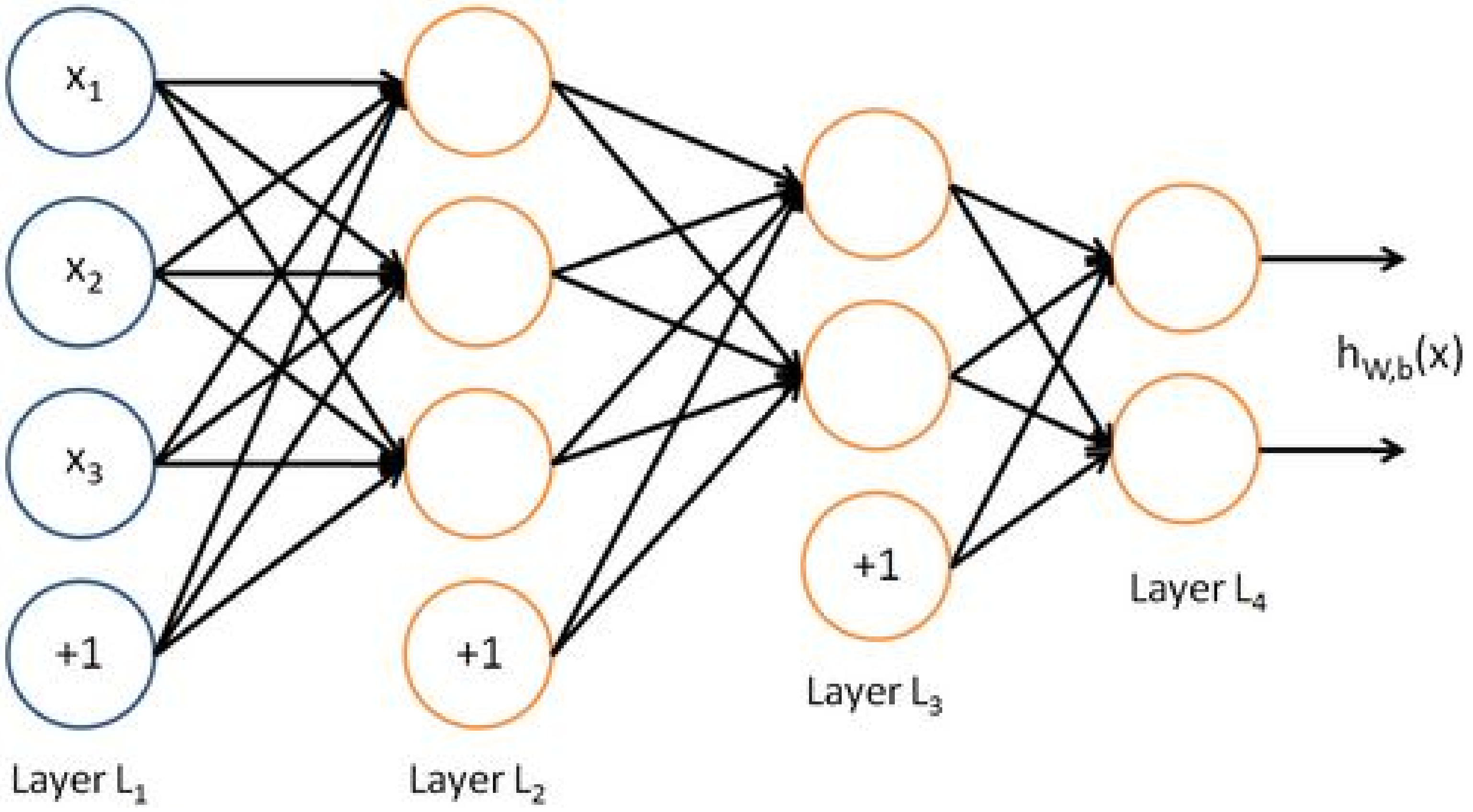


We can introduce a **bias factor** to help us with this

Artificial Neural Networks: Types

Bias Values

- Typically, a single bias node is added for the input layer and every hidden layer in a feedforward network. You would never add two or more to a given layer



Artificial Neural Networks: Types

Bias Values

- Essentially, bias values allow us to shift our activation functions from left to right to better fit the data
- Bias nodes are always on
- We can, in fact, initialize all of our bias nodes as the same value, say 0
- **Special Case:** For ReLU (We'll talk about this in a minute)some people like to use small constant value such as 0.01 for all biases because this ensures that all ReLU units fire in the beginning and therefore obtain and propagate some gradient

Activation Functions

Artificial Neural Networks: Types

Activation Functions

The purpose of the activation function is to introduce non-linearity into the network

ANNs are non linear models, and therefore we must conduct some form of nonlinear transformation

Without a non-linear activation function in the network, an ANN, no matter how many layers it had would behave just like a single perceptron

Artificial Neural Networks: Types

Activation Functions

The purpose of the activation function is to introduce non-linearity into the network

ANNs are non linear models, and therefore we must conduct some form of nonlinear transformation

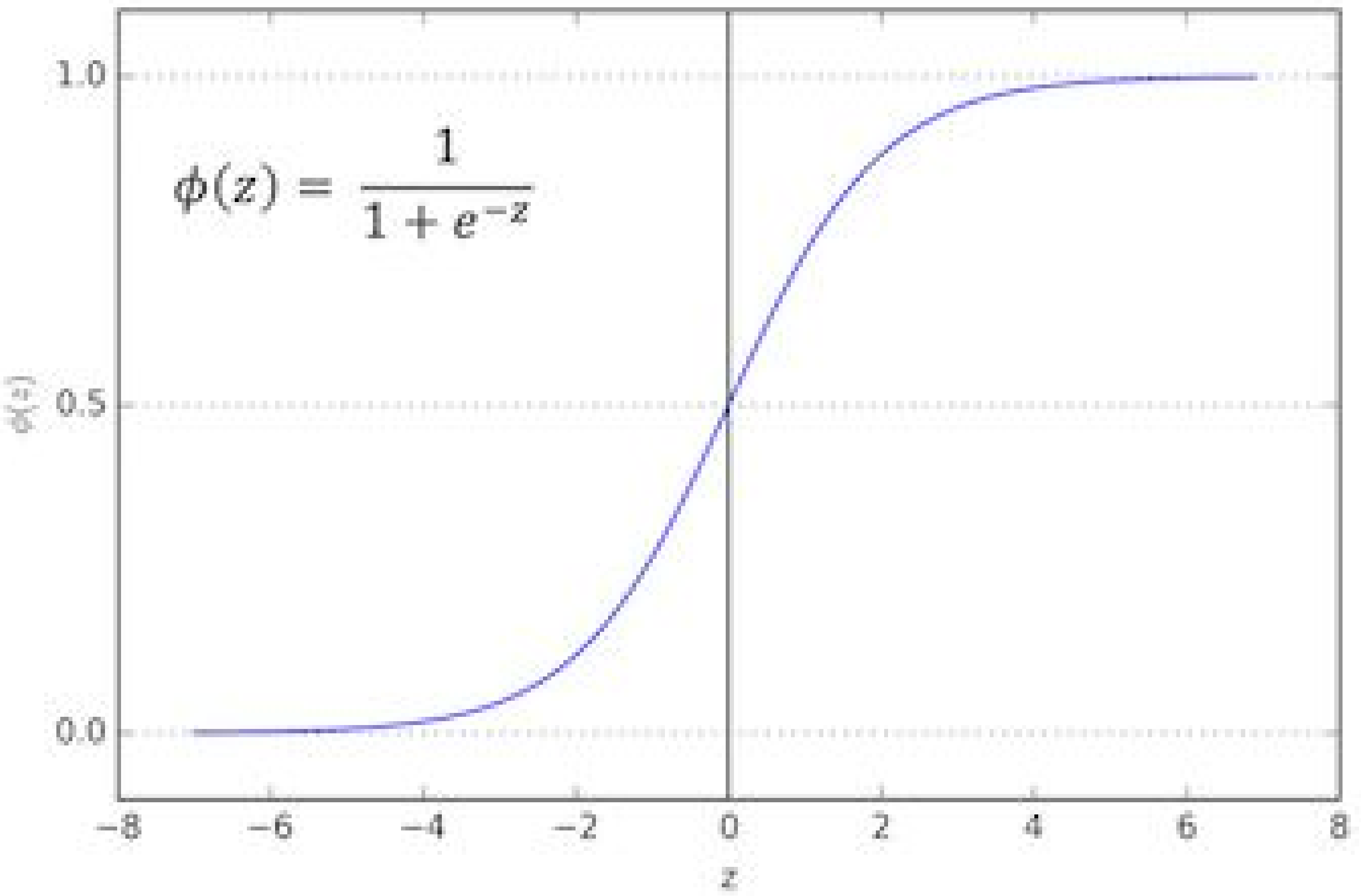
Without a non-linear activation function in the network, an ANN, no matter how many layers it had would behave just like a single perceptron

The activation function transforms our data so that our network can interpret it.

Artificial Neural Networks: Types

Activation Functions

You've seen activation functions before!

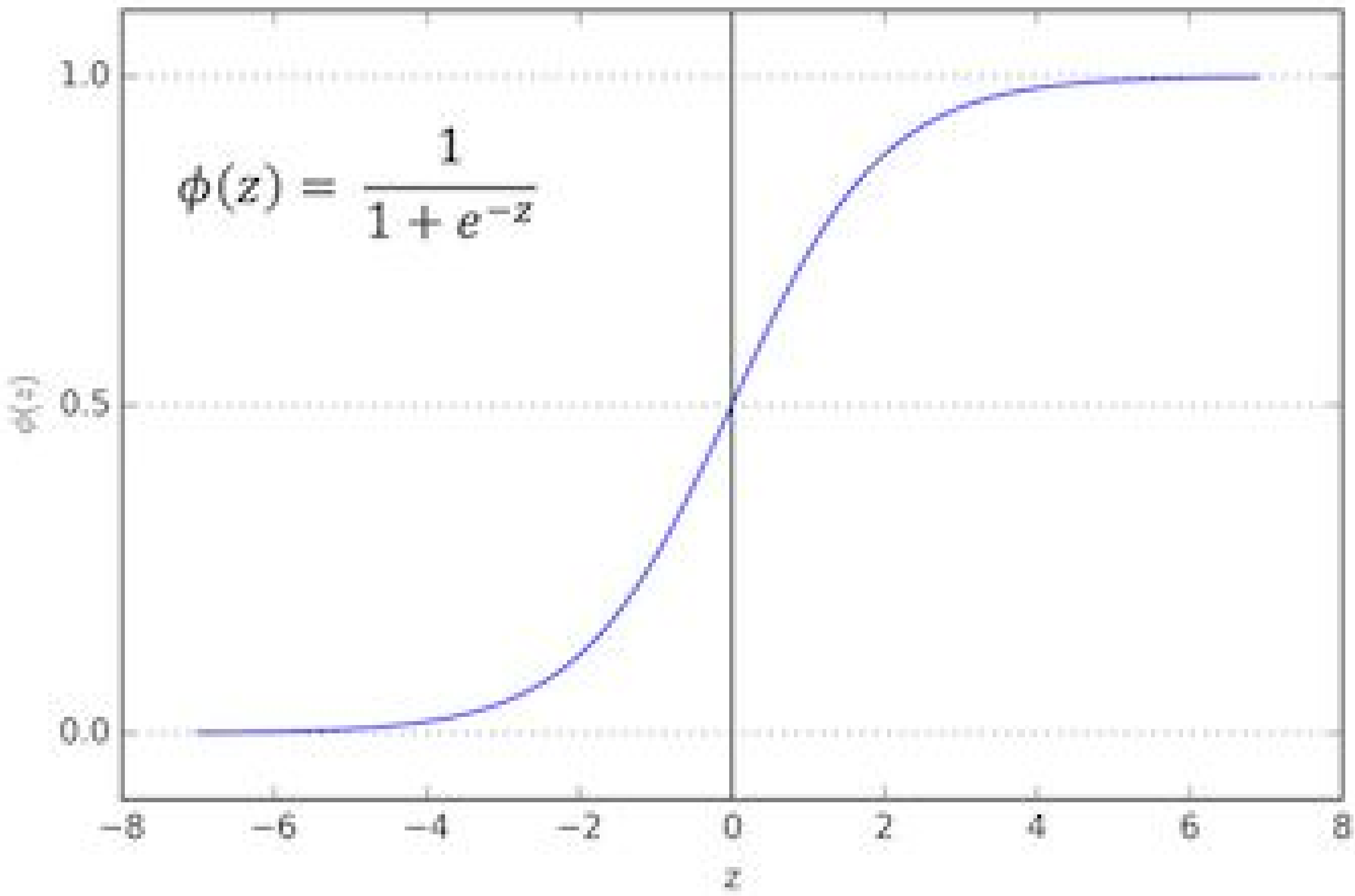


Artificial Neural Networks: Types

Activation Functions

You've seen activation functions before!

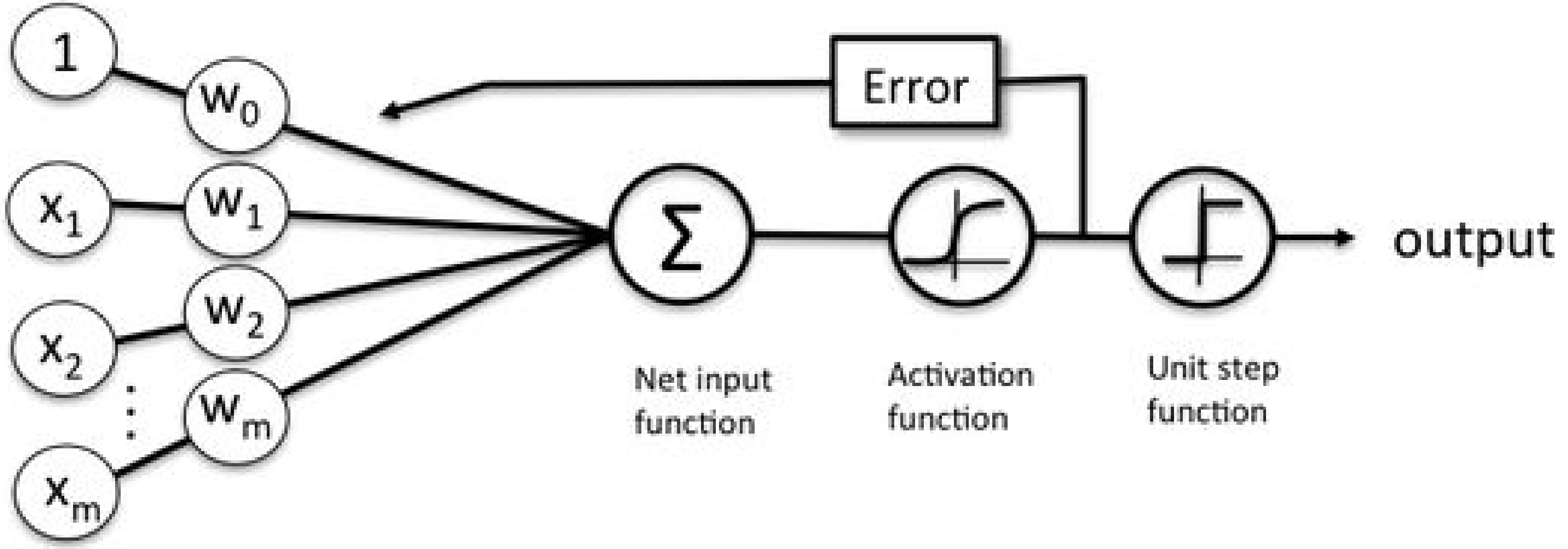
When conducting a logistic regression, your input goes through a nonlinear activation function, the logistic sigmoid function



Artificial Neural Networks: Types

Activation Functions

If we were to graph a logistic classifier, it would look conspicuously like a basic network



Schematic of a logistic regression classifier.

Artificial Neural Networks: Types

Activation Functions

A logistic classifier, however, still have linear based weights

A neural network, on the other hand, has non linear based weights

Artificial Neural Networks: Types

Activation Functions: Two Types

- The Sigmoid function

Sigmoid unit :

$$f(x) = \frac{1}{1+exp(-x)}$$

- ReLu

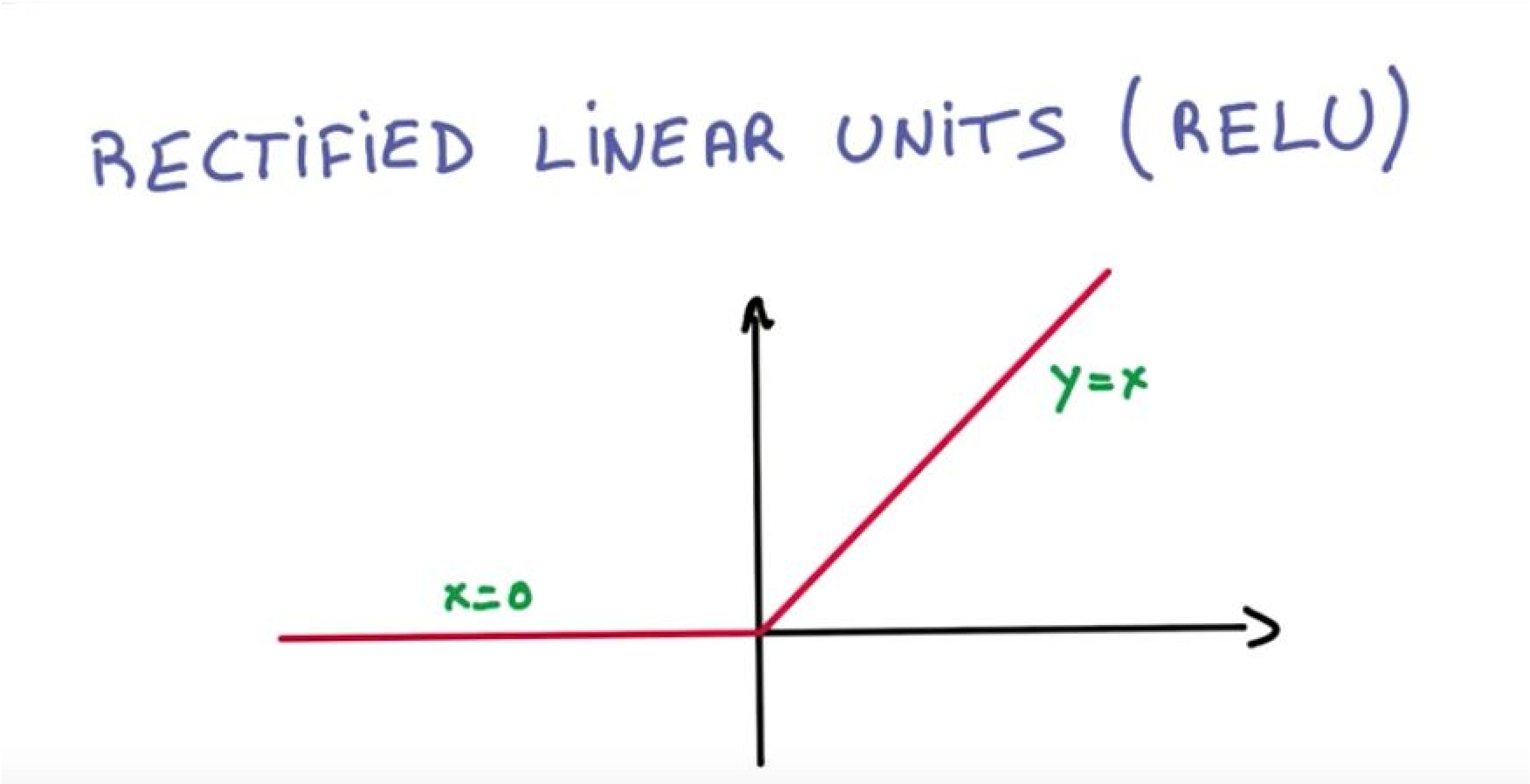
Rectified linear unit (ReLU):

$$f(x) = \sum_{i=1}^{\text{inf}} \sigma(x - i + 0.5) \approx log(1 + e^{x})$$

Artificial Neural Networks: Types

ReLu

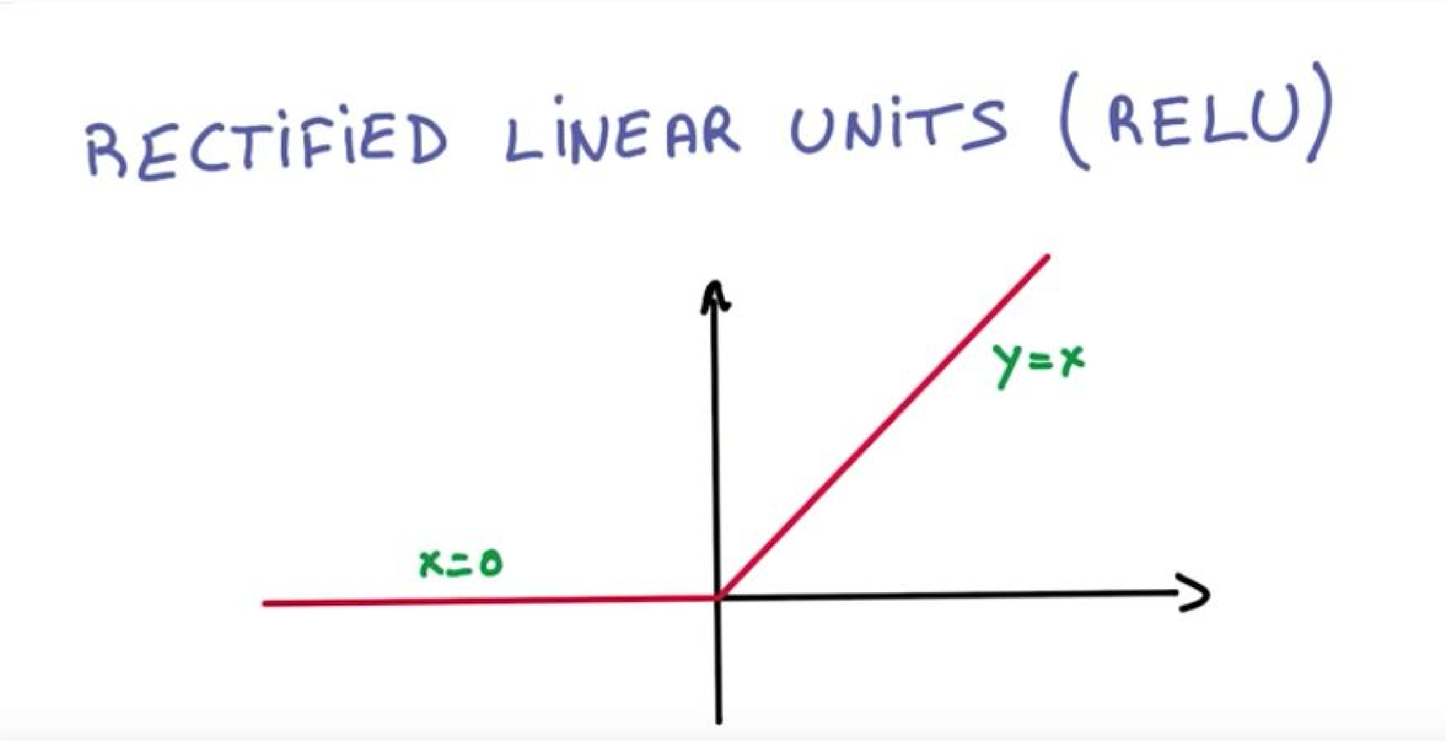
ReLu is perhaps the simplest non linear function



Artificial Neural Networks: Types

ReLu

They also have nice, simple derivatives



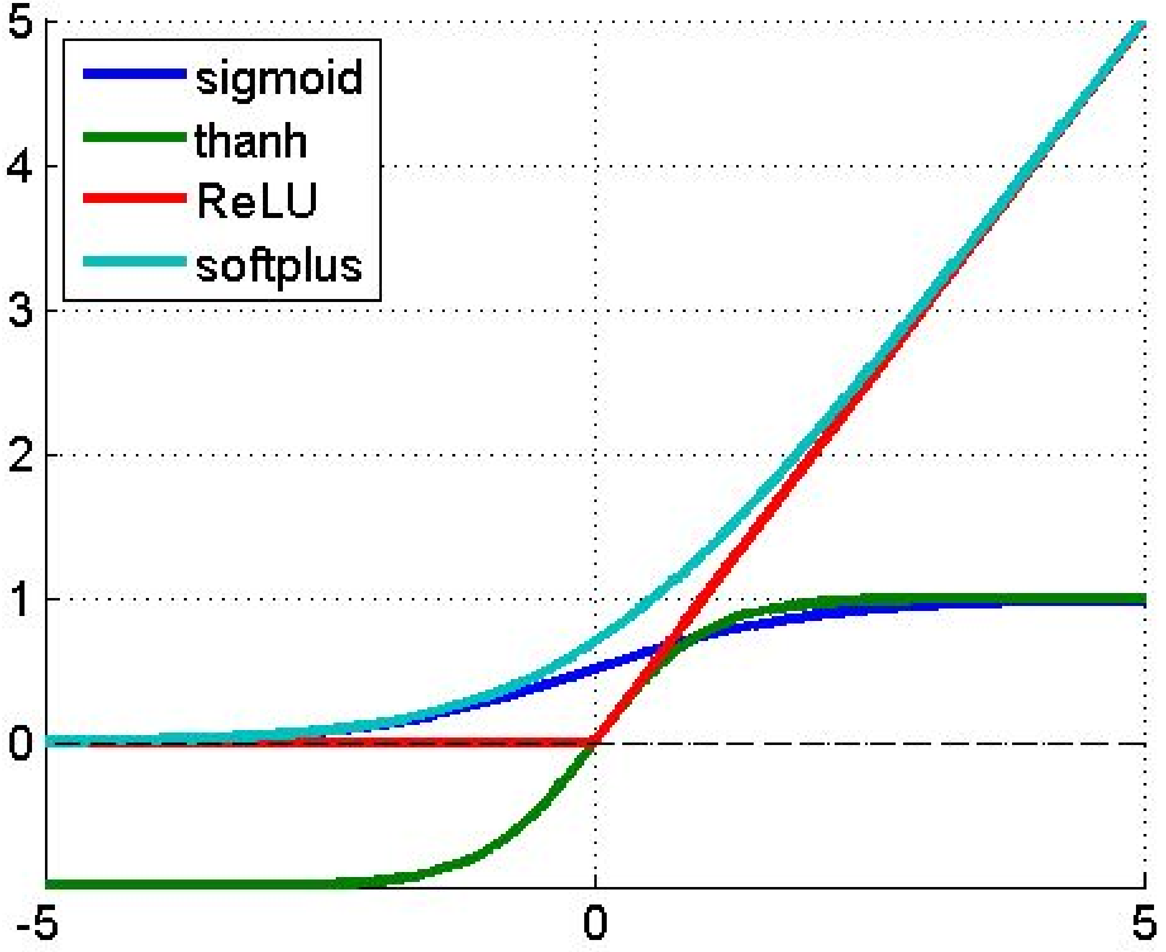
Artificial Neural Networks: Types

Activation Functions: Major Differences

- Sigmoid function has range [0,1] whereas the ReLu function has range [0,∞][0,∞]
- Given this, **Sigmoid is preferred for ANNs predicting probability**, while **ReLu is preferred for networks predicting positive, real numbers.**
- ReLu can help us solve the **vanishing gradient problem**, where networks with gradient based learning methods (traditional networks) have trouble updating the parameters of layers earlier in the network.

Artificial Neural Networks: Types

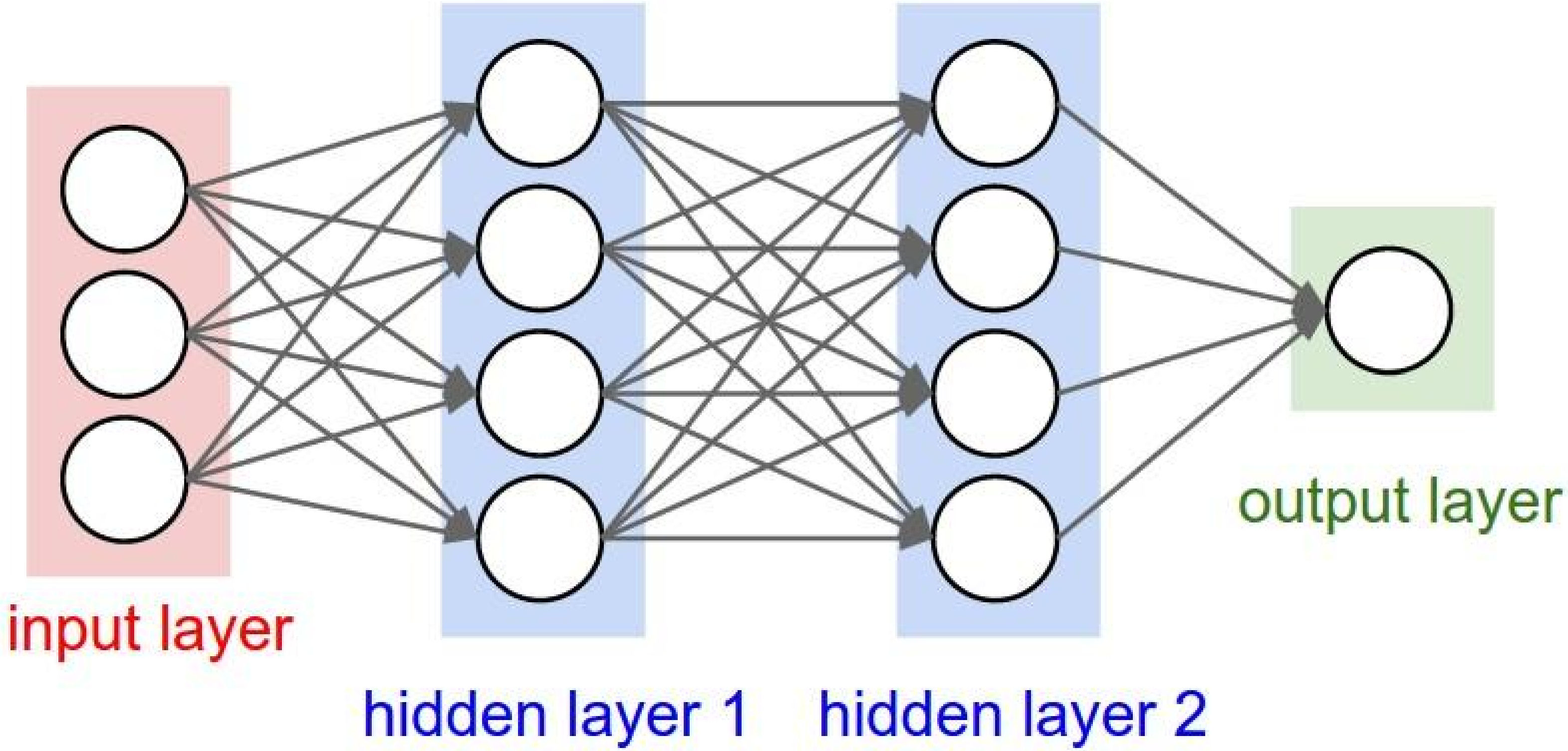
Activation Functions: Graphing activation functions



Artificial Neural Networks: Types

All Together Now: Forward Prop

The full forward pass of this 3-layer neural network is then simply three matrix multiplications, interwoven with the application of the activation function



Artificial Neural Networks: Types

All Together Now: Forward Prop

Pythonically, it looks like this:

```
# forward-pass of a 3-layer neural network:
f = lambda x: 1.0/(1.0 + np.exp(-x)) # activation function (use sigmoid)
x = np.random.randn(3, 1) # random input vector of three numbers (3x1)
h1 = f(np.dot(W1, x) + b1) # calculate first hidden layer activations (4x1)
h2 = f(np.dot(W2, h1) + b2) # calculate second hidden layer activations (4x1)
out = np.dot(W3, h2) + b3 # output neuron (1x1)
```

Source: Harvard University

Cost Functions

Artificial Neural Networks: Cost Functions

There are two primary cost functions with ANNs

- Squared Error Cost Function (Sometimes called quadratic cost)

$$E_{total} = \frac{1}{2}\sum^{N}(target_i - output_i)^2$$

- Cross Entropy Cost Function

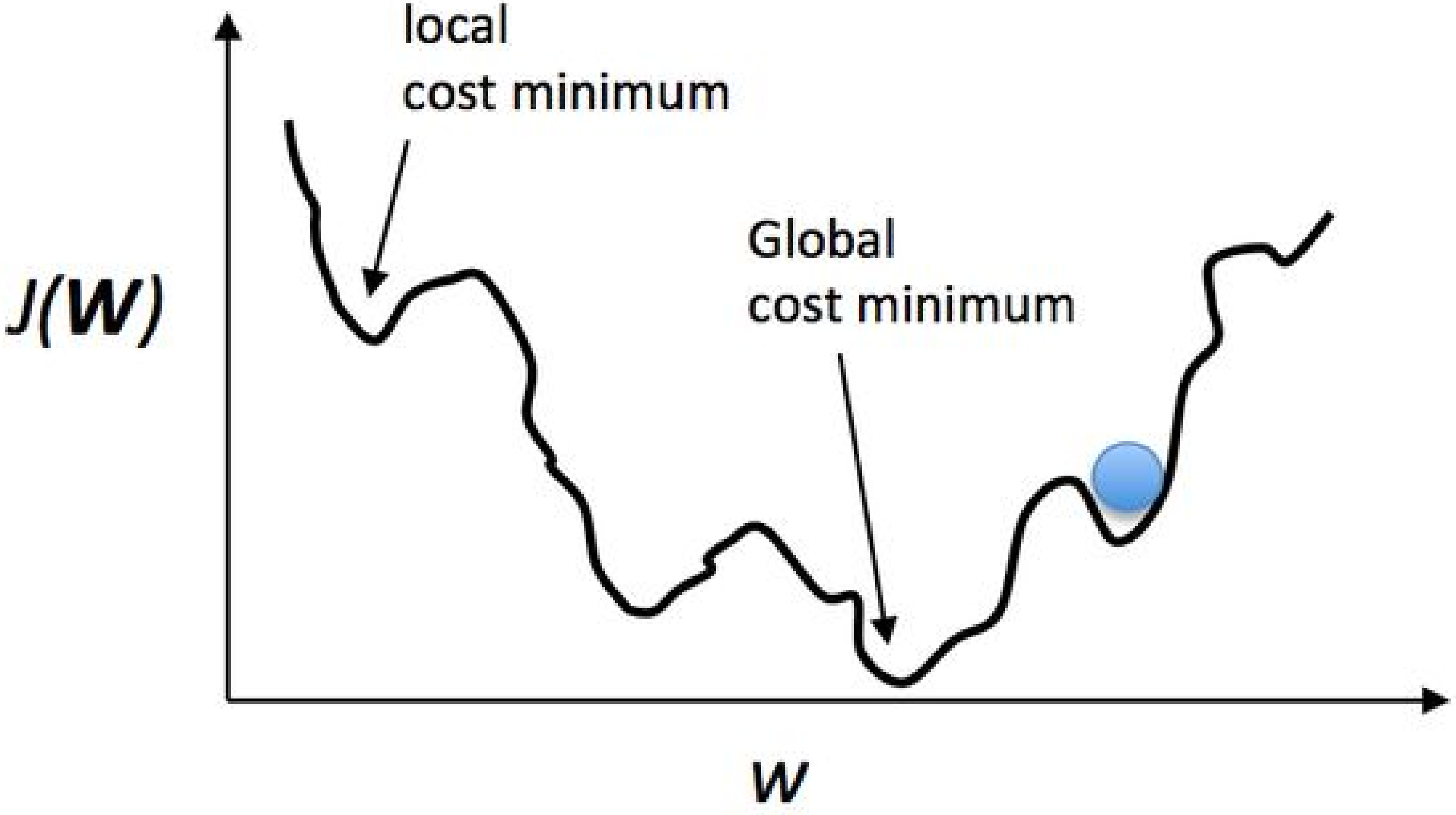
$$-\sum_{i=0}^{n}\ln(o_i) * t_i$$

Stochastic Gradient Descent

Artificial Neural Networks: Types

Backpropogation

Recall our friend gradient descent



Artificial Neural Networks: Types

Stochastic Gradient Descent

Imagine calculating gradient descent for a large amount of features - how computationally expensive would this be?

- When we calculate our loss function; calculating gradient descent to minimize this loss functions takes **<u>three times as long</u>**
- So how do we solve this?
  - We compute a ***very bad estimate of the loss***, which will be the average loss for a very small **random** section of the training data
  - We'll do this ***many, many times*** to eventually reach the minima

This is what we call Stochastic Gradient Descent

Artificial Neural Networks: Types

Stochastic Gradient Descent

Suppose we have weights w and biases b in our neural network. Then stochastic gradient descent works by picking out a randomly chosen mini-batch of training inputs, and training with those:

$$w_k \rightarrow w'_k = w_k - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial w_k}$$

$$b_l \rightarrow b'_l = b_l - \frac{\eta}{m} \sum_j \frac{\partial C_{X_j}}{\partial b_l},$$

Source: neuralnetworksanddeeplearning.com

where the sums are over all the training examples Xj in the current mini-batch.

Then we pick out another randomly chosen mini-batch and train with those and repeat, until we've exhausted the training inputs, which is said to complete an e**poch of training.**

Artificial Neural Networks: Types

Stochastic Gradient Descent

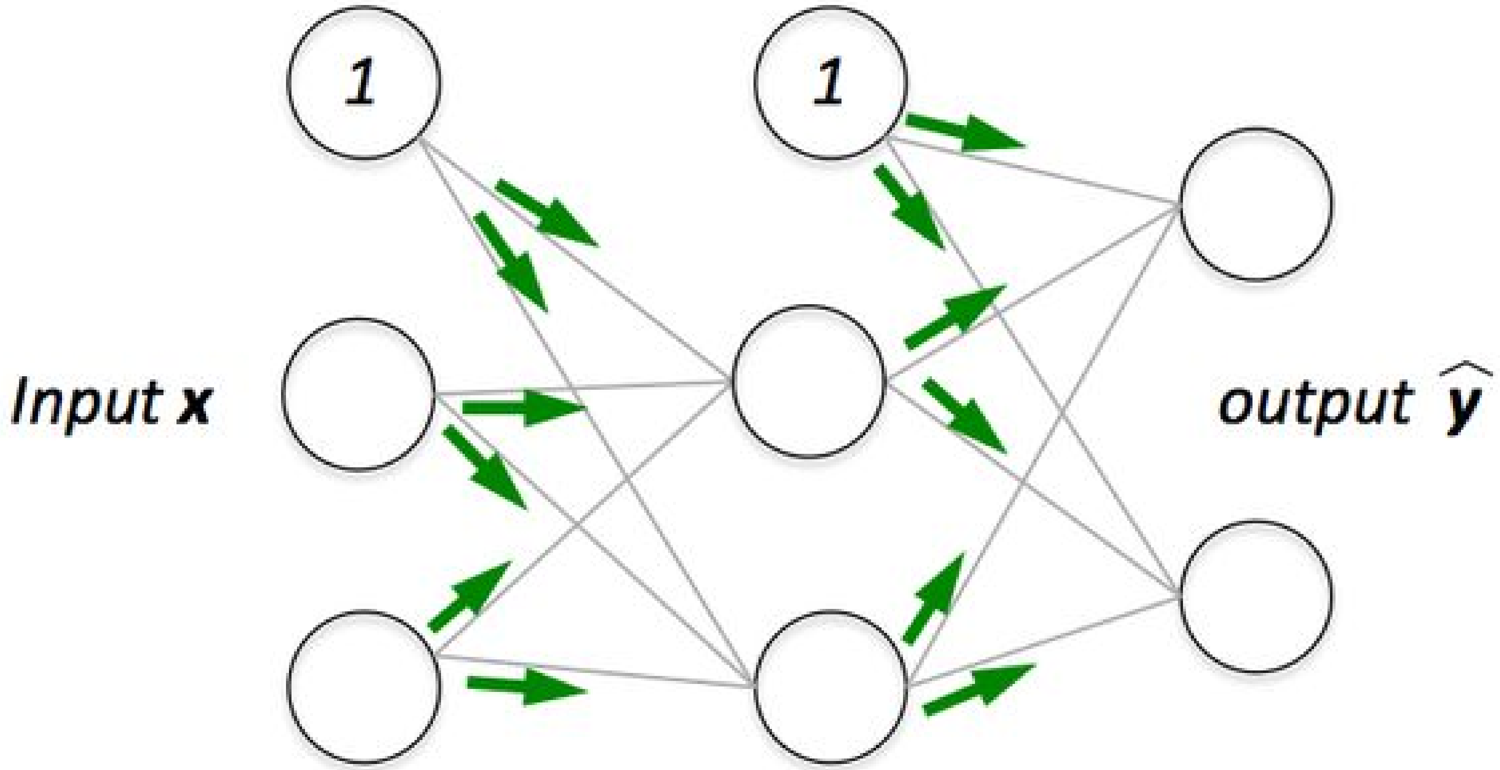
- Calculating gradient descent for a neural network with respect to its weights can get tricky. It’s not just taking a partial derivative, it’s taking it with respect to the many nested inner functions that neural networks have.
- We can solve this problem using a technique called **backpropagation**

Backpropogation

Artificial Neural Networks: Types

Backpropagation

Think about how information flows through a network; we call this forward propagation

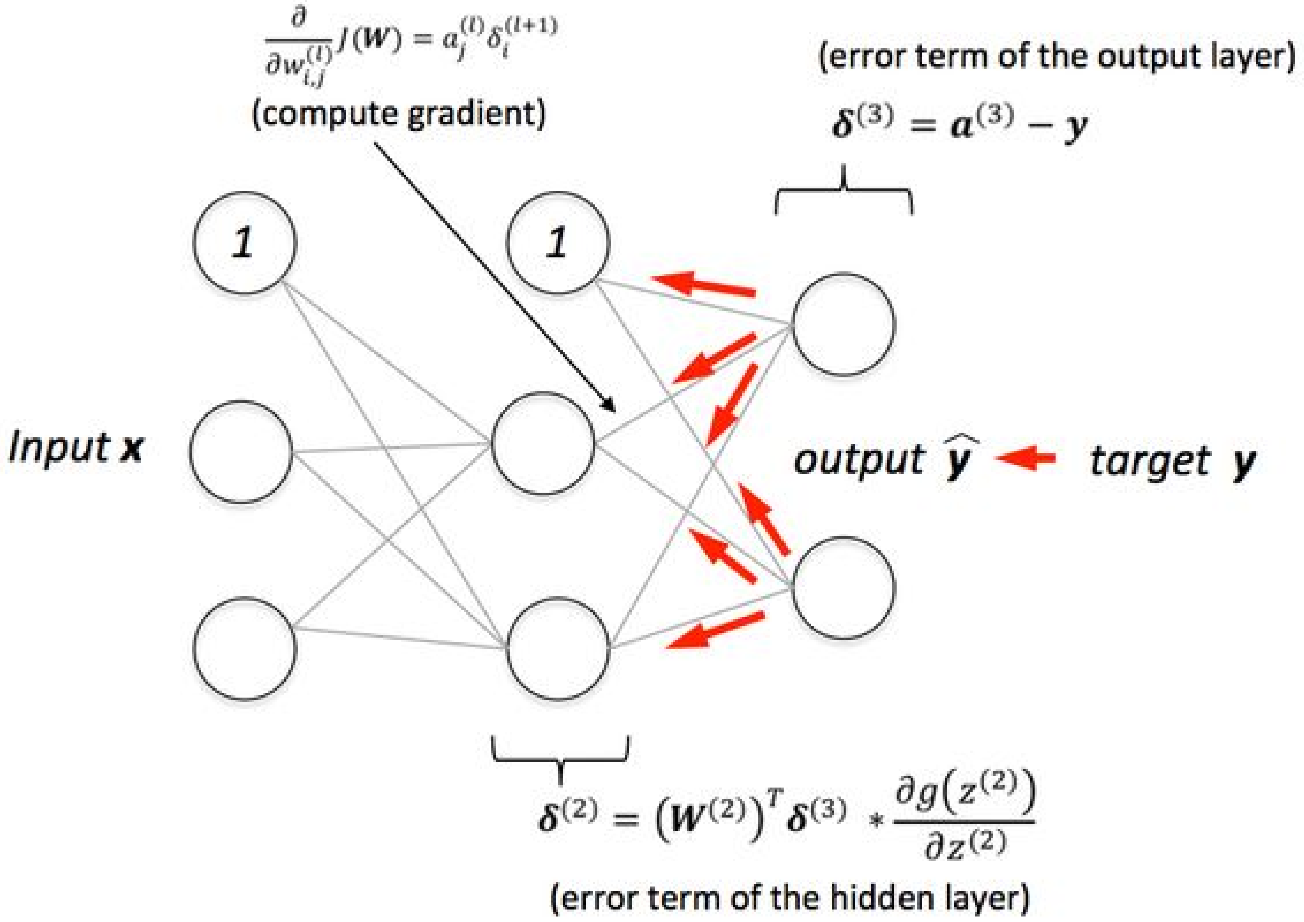


Artificial Neural Networks: Types

Backpropagation

At the end of this classification, we will have error (the "cost" that you compute by comparing the calculated output and the known, correct target output). We can send this error **back** through the network in a process called **backpropagation**.

This is the workhorse of learning in ANNS



Source: Sebastian Raschka

Artificial Neural Networks: Types

Backpropagation

- Backprop helps us optimize weights for our network
- Helps us compute the derivative of complex functions when performing gradient descent

Artificial Neural Networks: Types

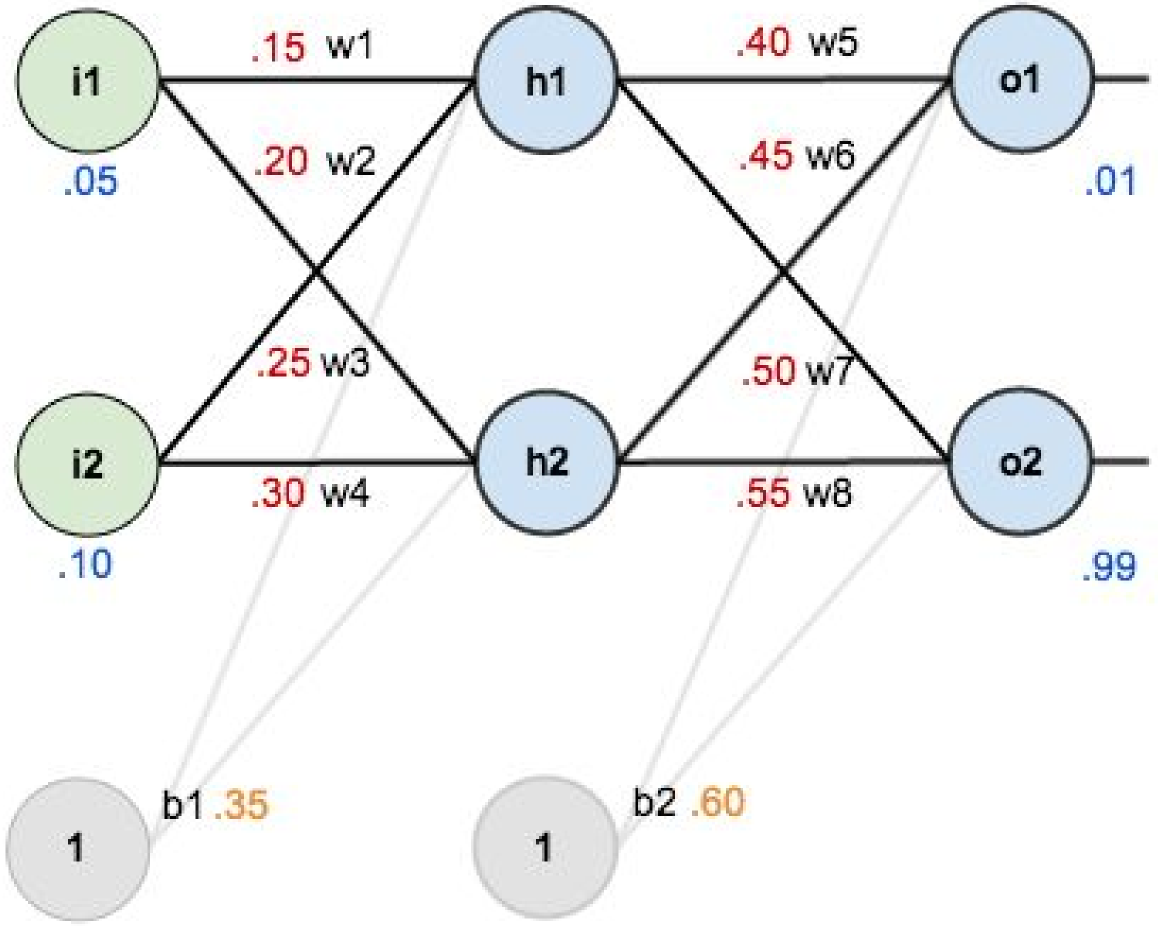
Backpropagation

Let's see backpropagation in action so we can understand it, as well as the learning process

Artificial Neural Networks: Putting it all together

For ANNs, we can think of a general process:

1. We figure out the total net input to each hidden layer neuron
2. Squash the total net input using an activation function
3. Repeat the process with the output layer neurons.

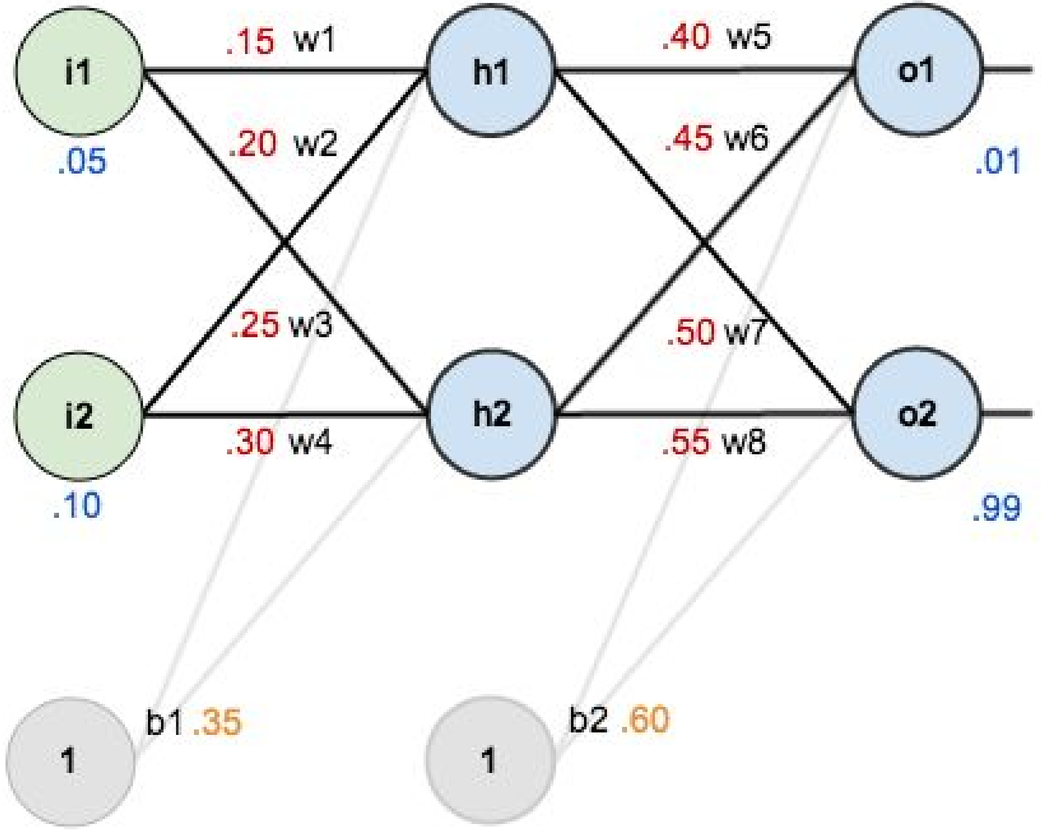


Artificial Neural Networks: Putting it all together

In this example, each net input would be calculated as:

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$



Artificial Neural Networks: Putting it all together

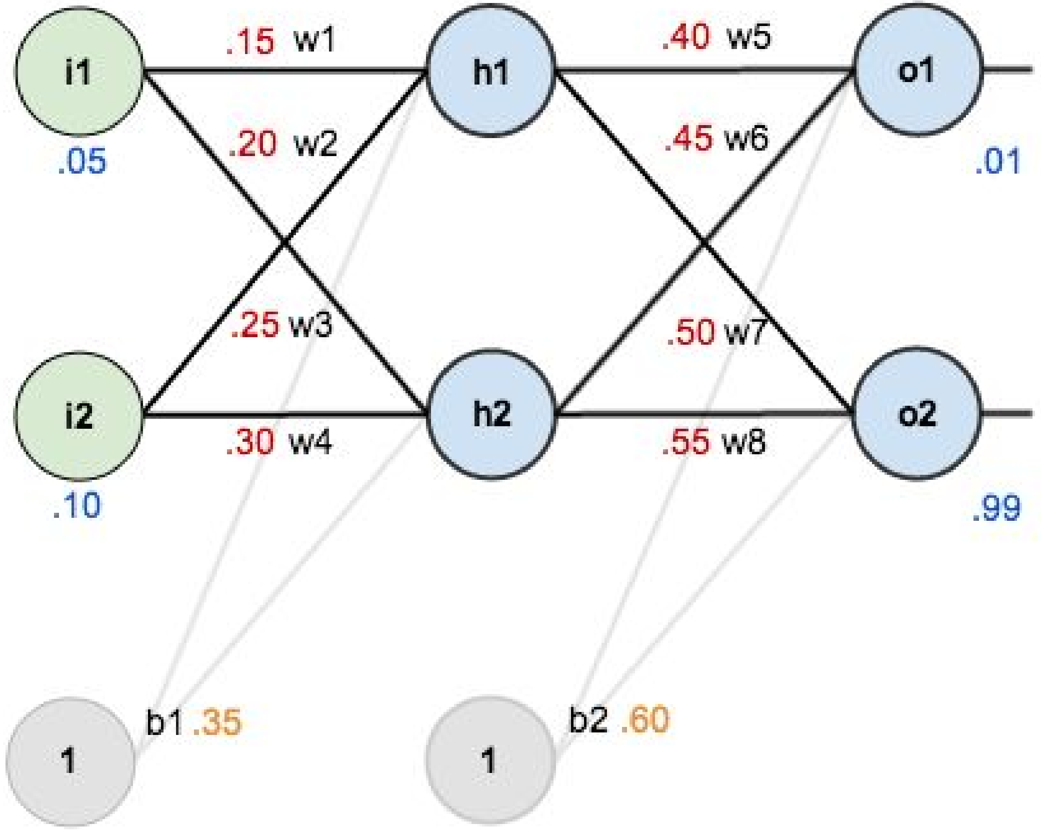
In this example, each net input would be calculated as:

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

We than want to put it through our Activation function, in this case:

$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$$



Artificial Neural Networks: Putting it all together

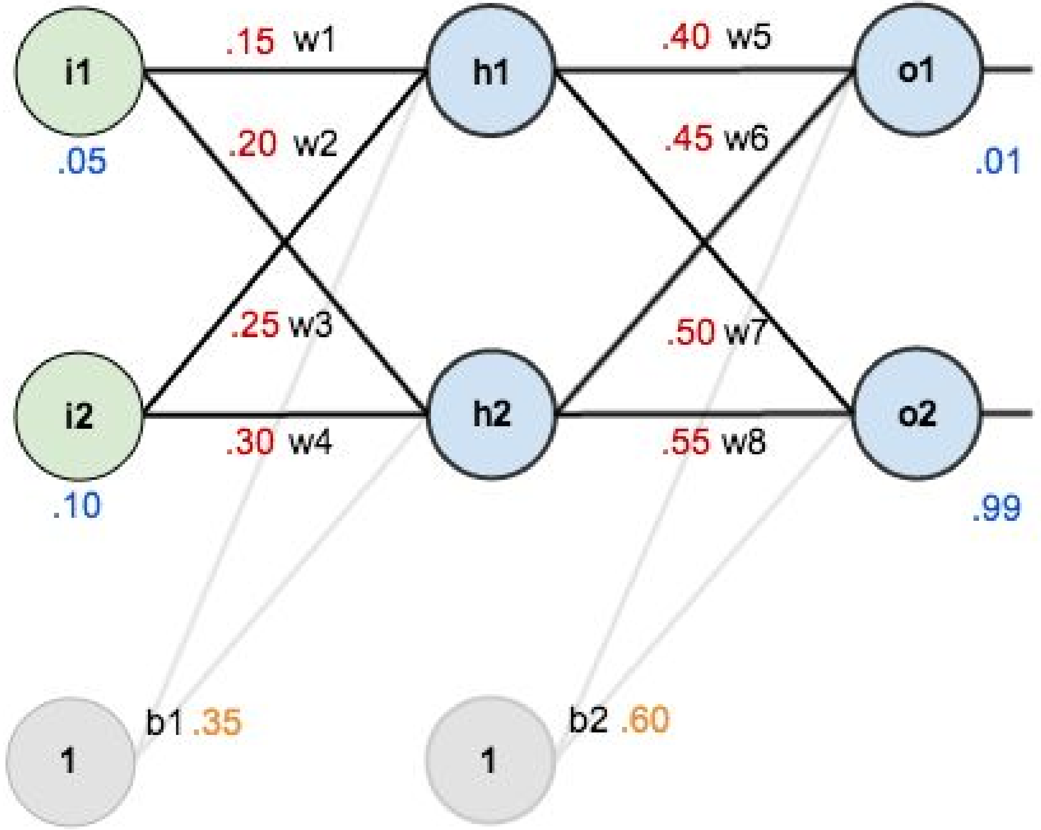
In this example, each net input would be calculated as:

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

We than want to put it through our Activation function, in this case:

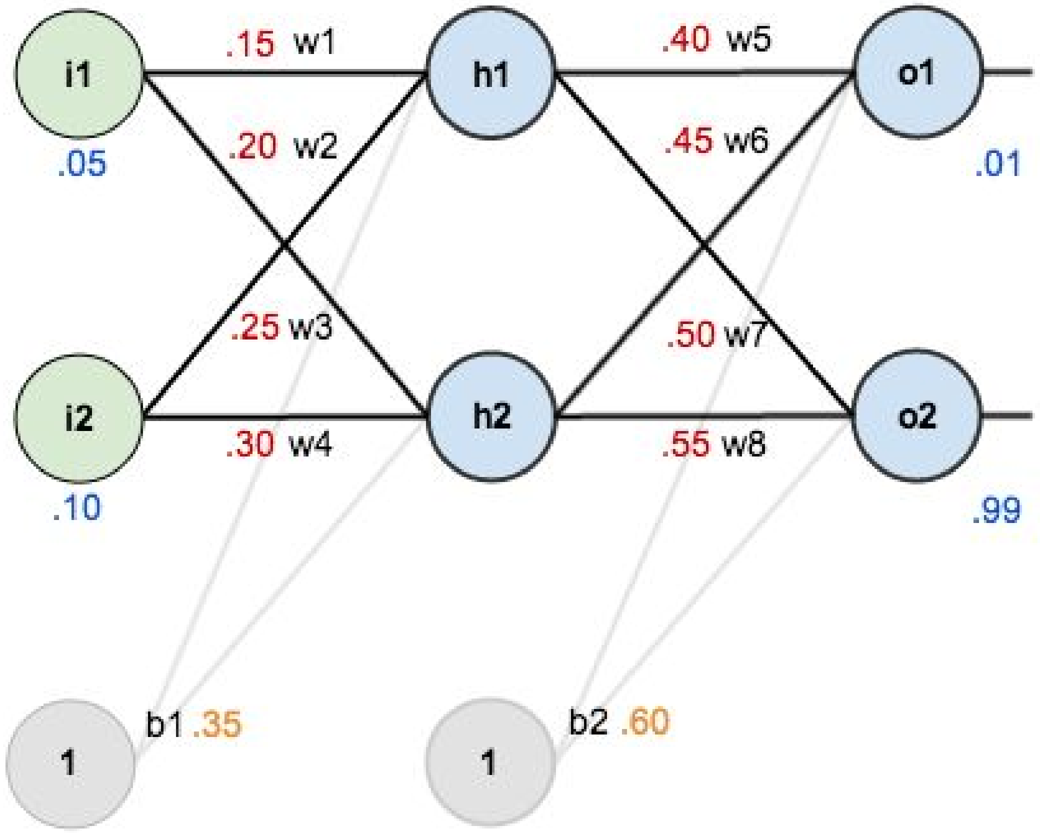
$$out_{h1} = \frac{1}{1+e^{-net_{h1}}} = \frac{1}{1+e^{-0.3775}} = 0.593269992$$



Artificial Neural Networks: Putting it all together

We'd do this process for each of the hidden layer neurons.

We then repeat this process for the output layer neurons, using the output from the hidden layer neurons as inputs.



$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$$

$$out_{o1} = \frac{1}{1+e^{-net_{o1}}} = \frac{1}{1+e^{-1.105905967}} = 0.75136507$$

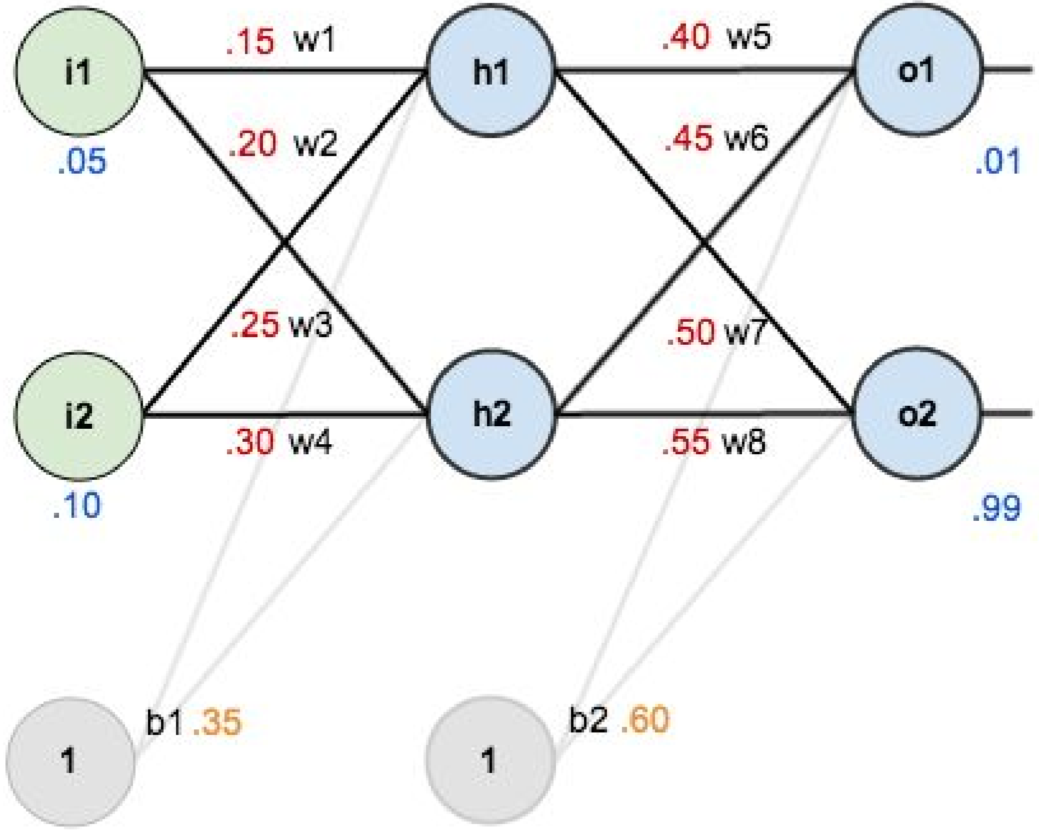
Artificial Neural Networks: Putting it all together

At the end of this process, we have our error.

For this example, we're going to be looking at the most basic error function, **the squared error function**.

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

The 1/2 is included so that exponent is cancelled when we differentiate later on



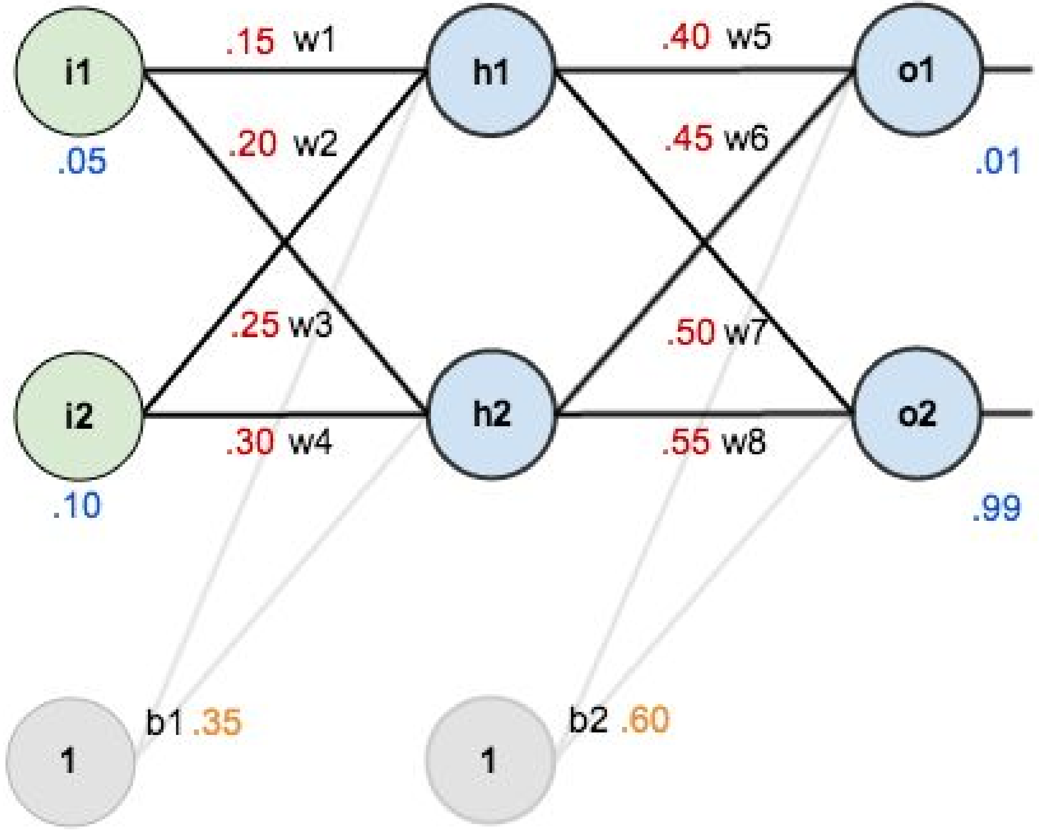
Artificial Neural Networks: Putting it all together

The error for output neuron 1 would then be:

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

=

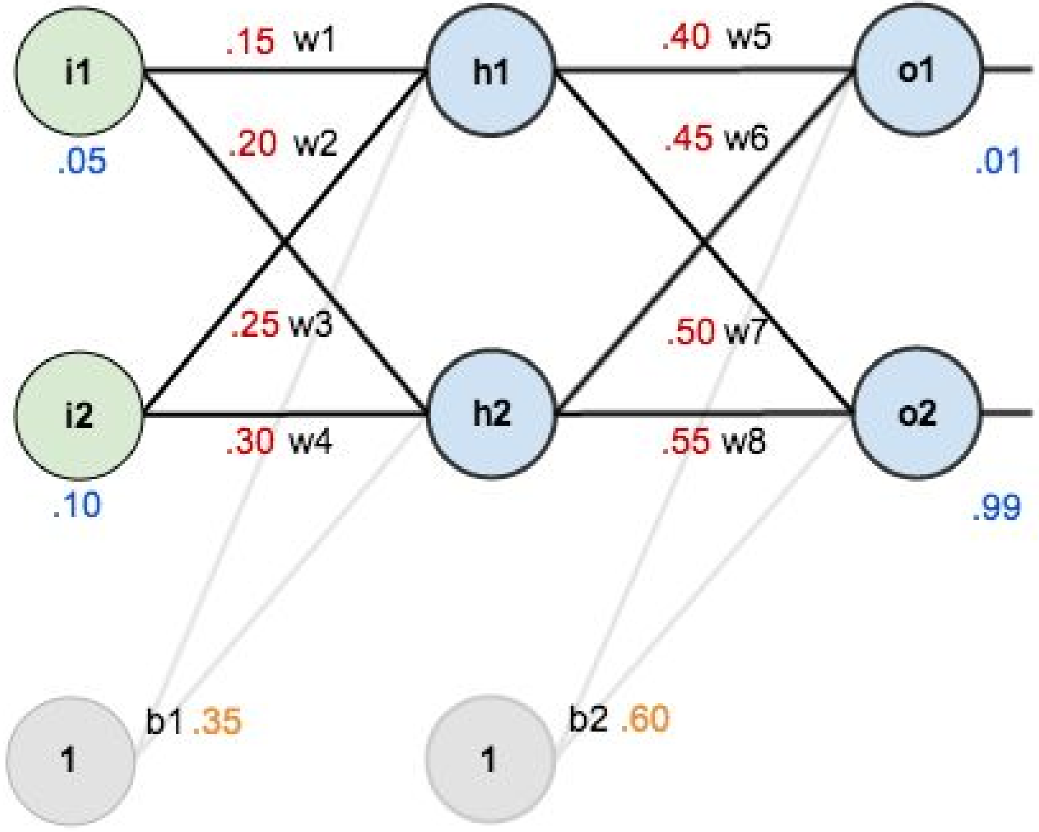
$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2 = \frac{1}{2}(0.01 - 0.75136507)^2 = 0.274811083$$



Artificial Neural Networks: Putting it all together

Now, let's do backpropagation!

Our goal with backpropagation is to update each of the weights in the network so that they cause the actual output to be closer the target output, thereby minimizing the error for each output neuron and the network as a whole.

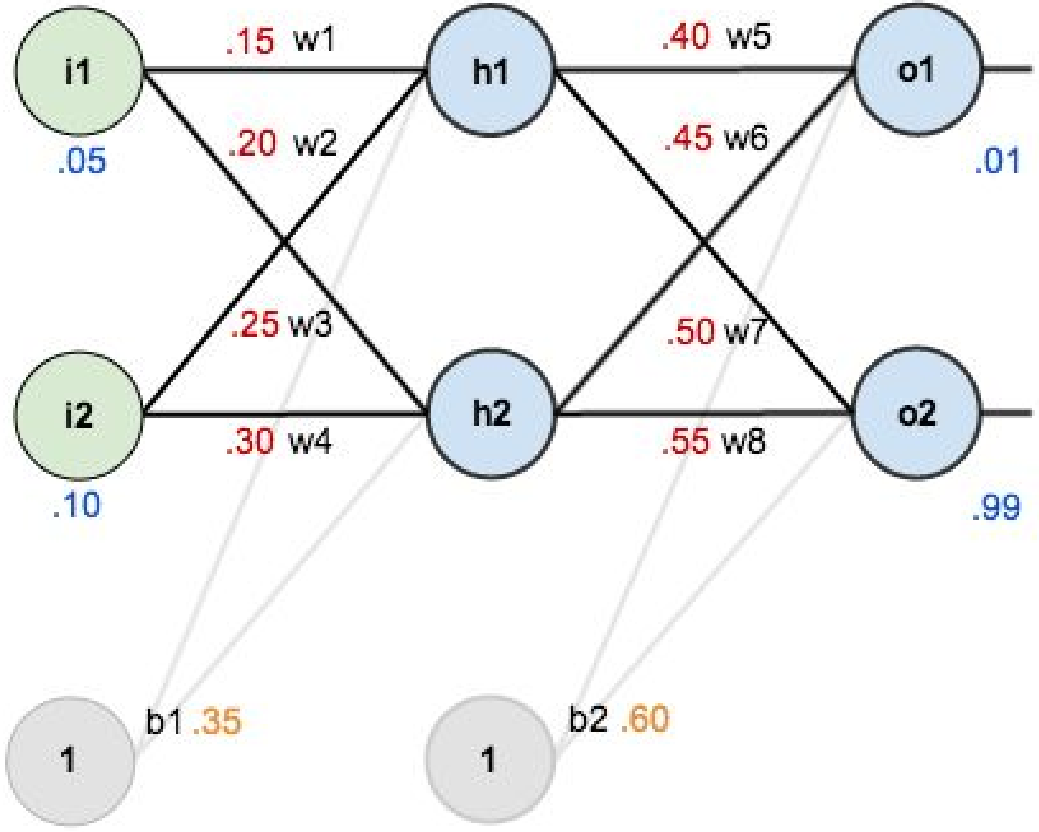


Artificial Neural Networks: Putting it all together

Now, let's do backpropagation!

Our goal with backpropagation is to update each of the weights in the network so that they cause the actual output to be closer the target output, thereby minimizing the error for each output neuron and the network as a whole.

Consider w_5. We want to know how much a change in w_5 affects the total error $\frac{\partial E_{total}}{\partial w_5}$.



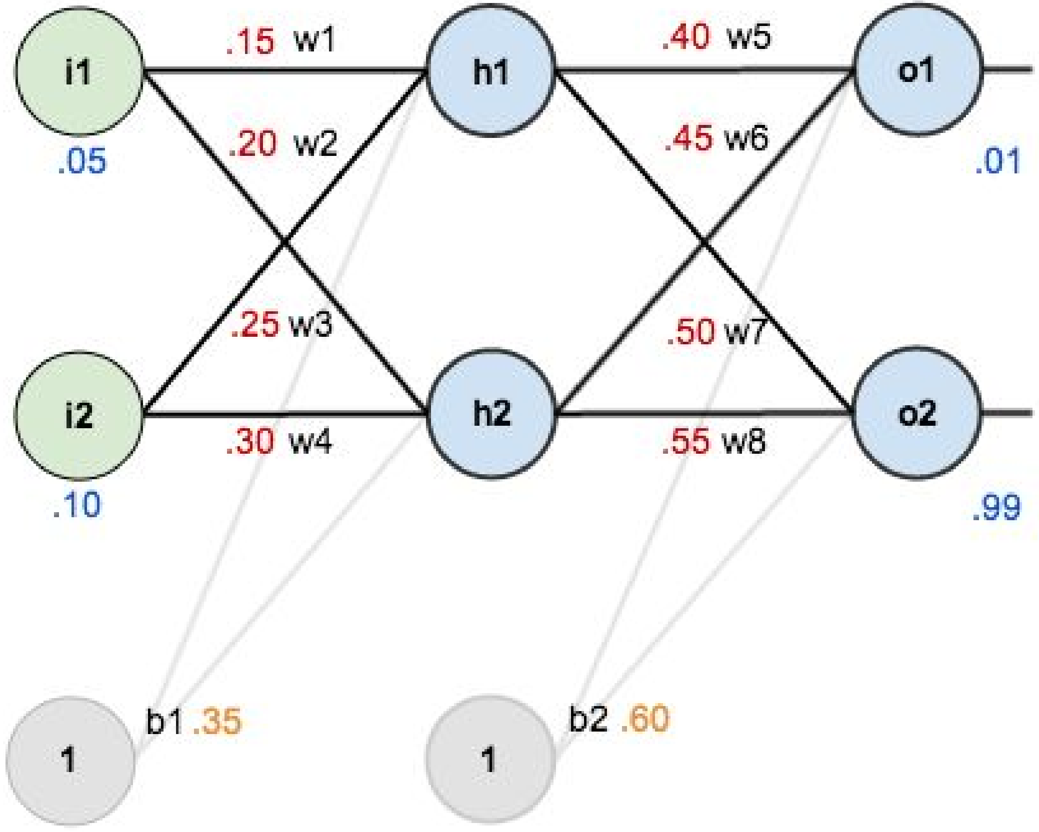
Artificial Neural Networks: Putting it all together

It’s back to high school calc:

We can solve this with the chain rule

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

And solve for each on the right side of the equation



Artificial Neural Networks: Putting it all together

It's back to high school calc:

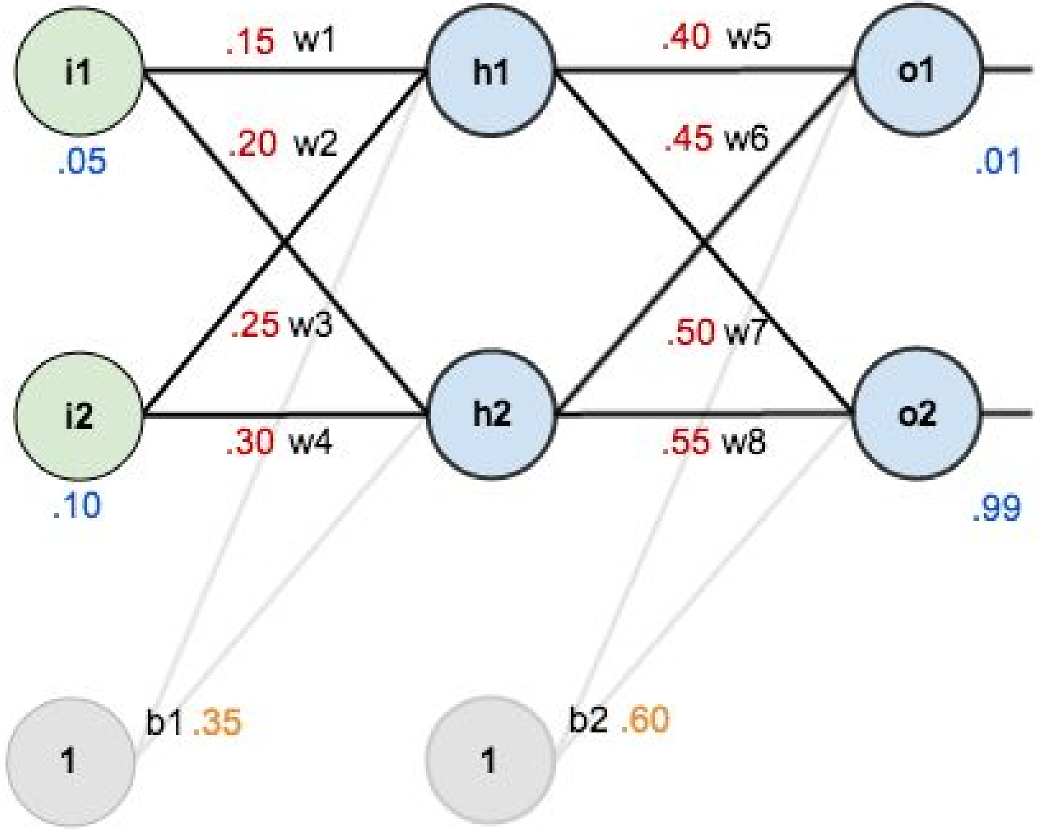
We can solve this with the chain rule

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial net_{o1}}{\partial w_5}$$

Finally, how much does the total net input of o1 change with respect to w_5?

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{o1}}{\partial w_5} = 1 * out_{h1} * w_5^{(1-1)} + 0 + 0 = out_{h1} = 0.593269992$$



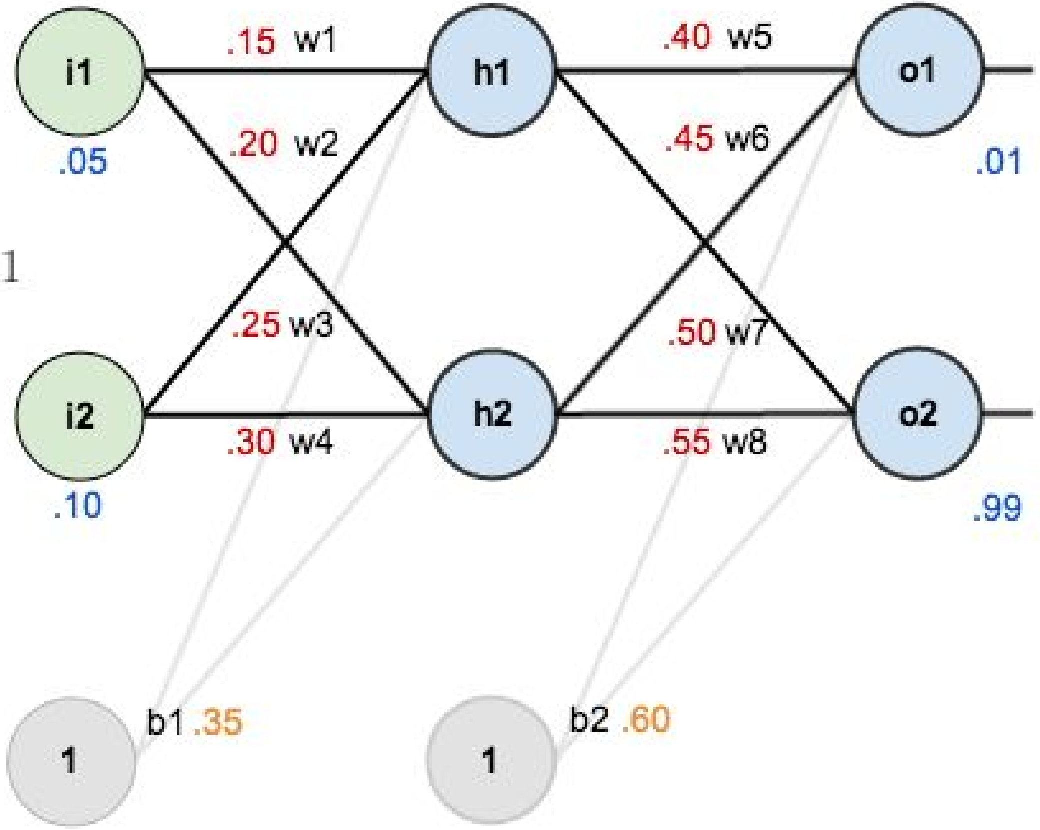
Artificial Neural Networks: Putting it all together

This gives us:

$$\frac{\partial E_{total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$

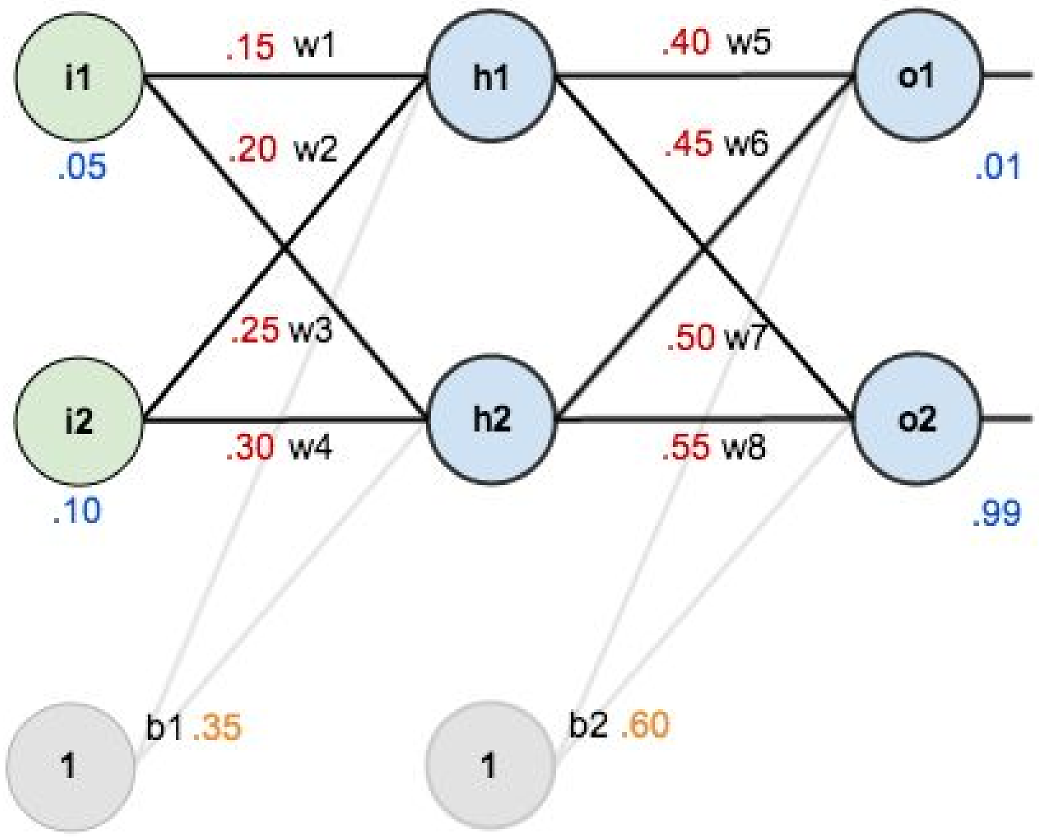
To decrease the error, we then subtract this value from the current weight (optionally multiplied by the learning rate)

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$



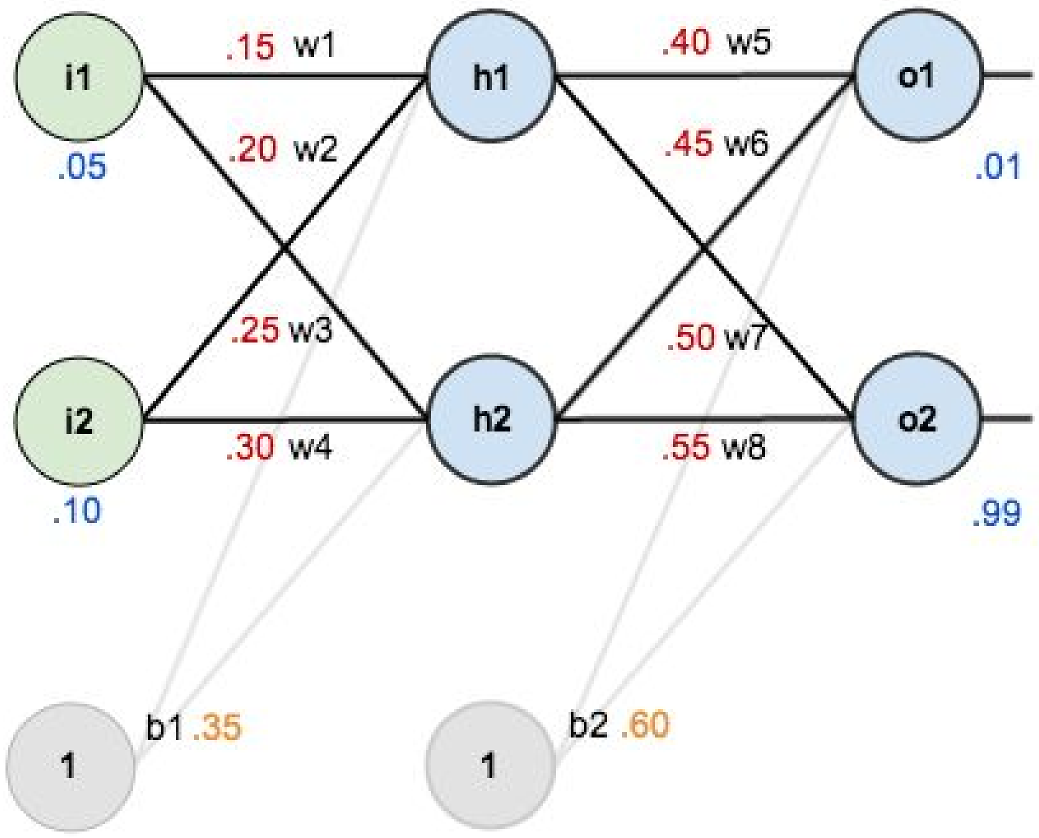
Artificial Neural Networks: Putting it all together

This process then repeats for the other weights in this layer.



Artificial Neural Networks: Putting it all together

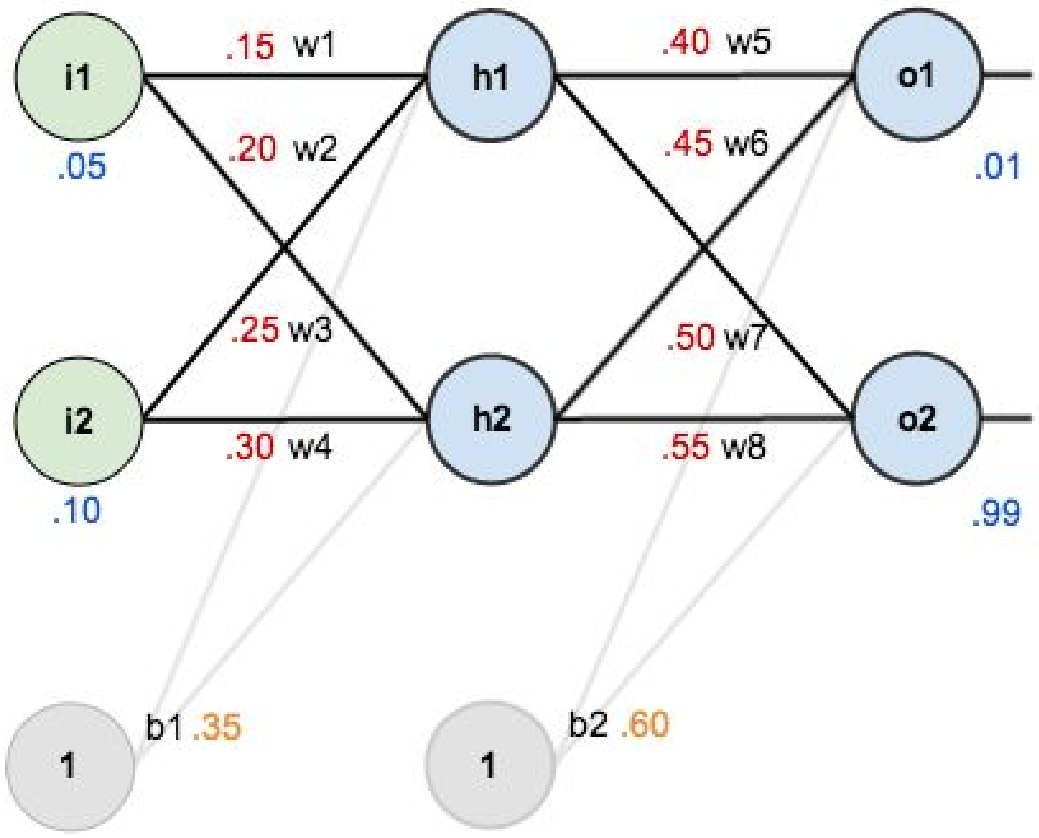
But wait……what about the weights for the hidden layer?



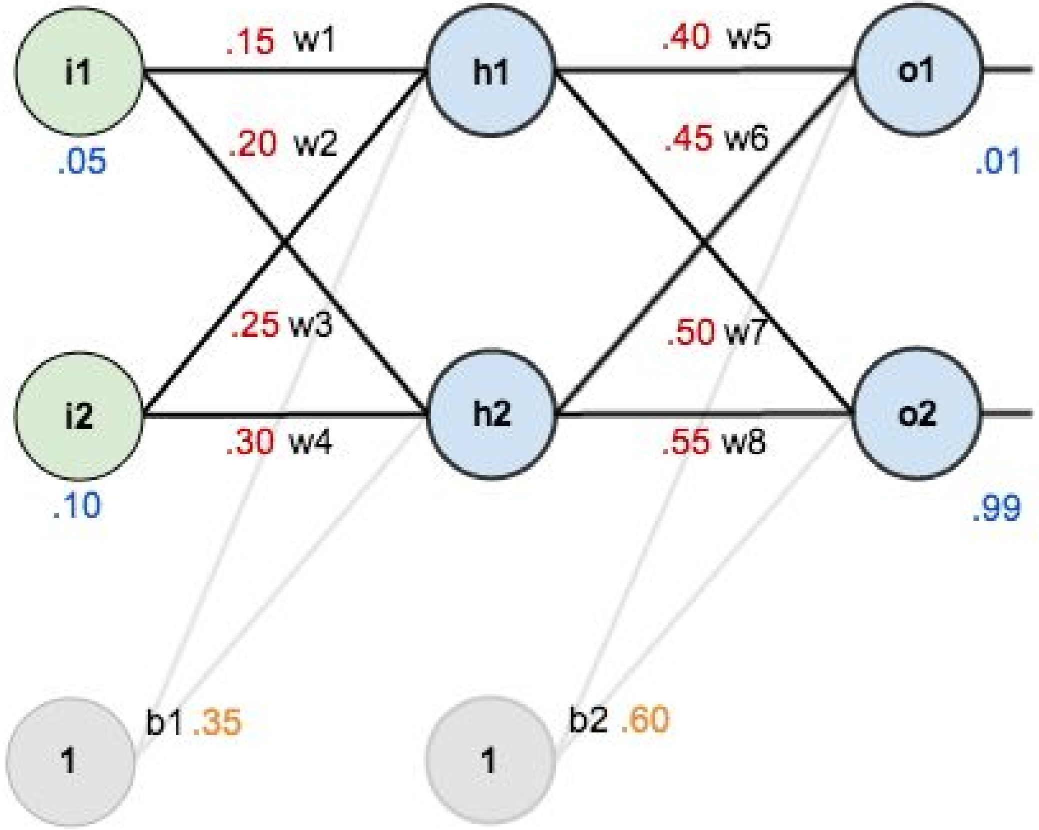
Artificial Neural Networks: Putting it all together

But wait......what about the weights for the hidden layer?

There's more



Artificial Neural Networks: Putting it all together

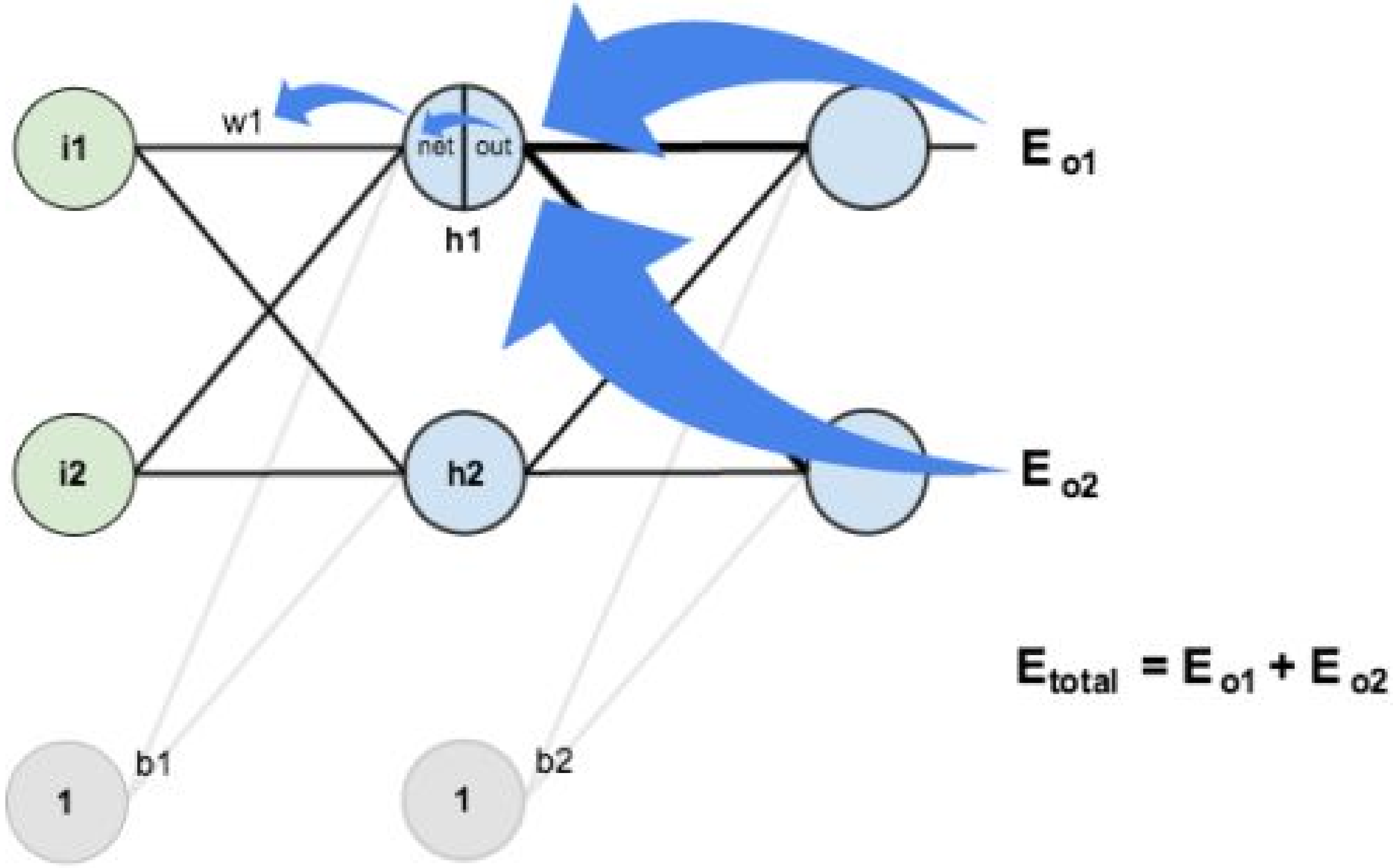


Artificial Neural Networks: Putting it all together

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\downarrow$$

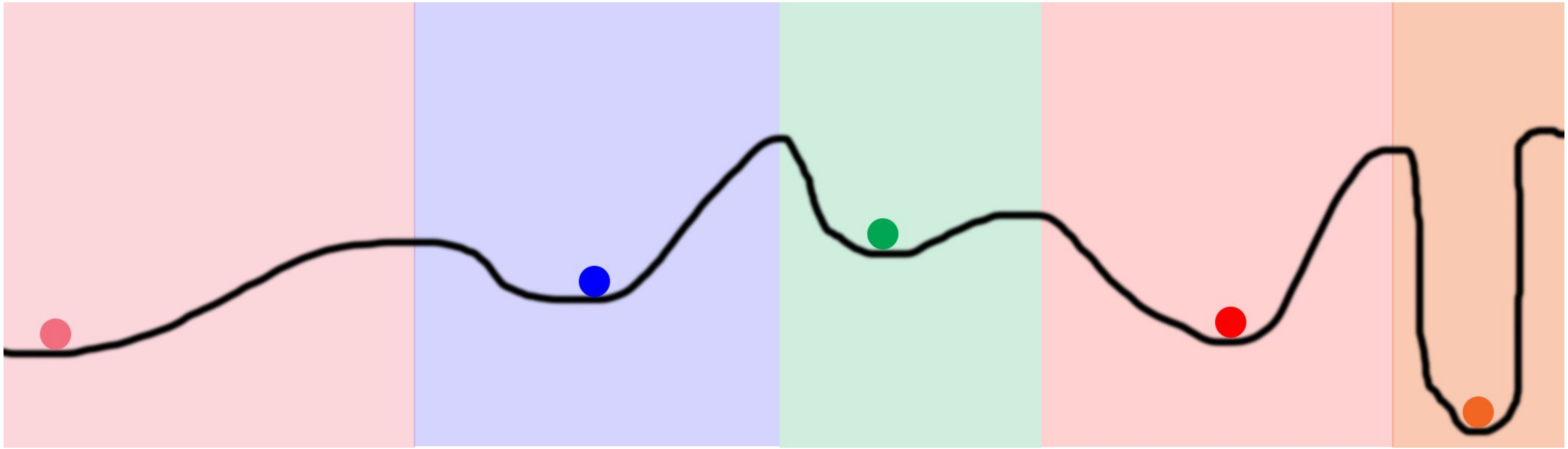
$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



Artificial Neural Networks: Types

Dropout (This is pretty cutting edge stuff)

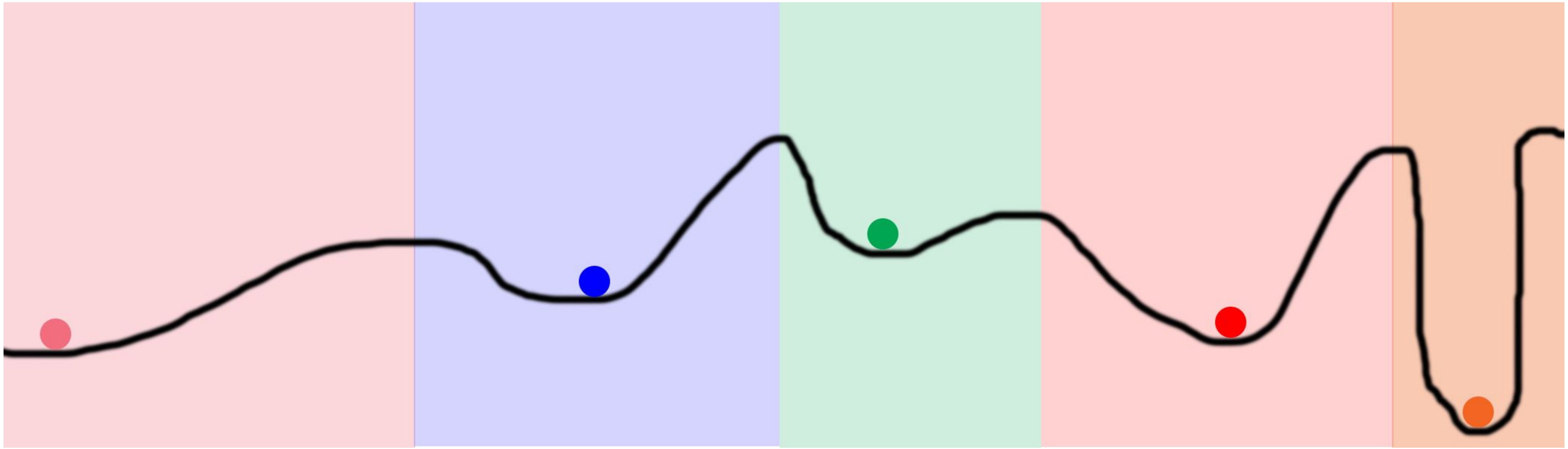
- Dropout is a simple way to prevent ANNs from overfitting that was developed in 2012.
- Consider the following examples below:



Artificial Neural Networks: Types

Dropout (This is pretty cutting edge stuff)

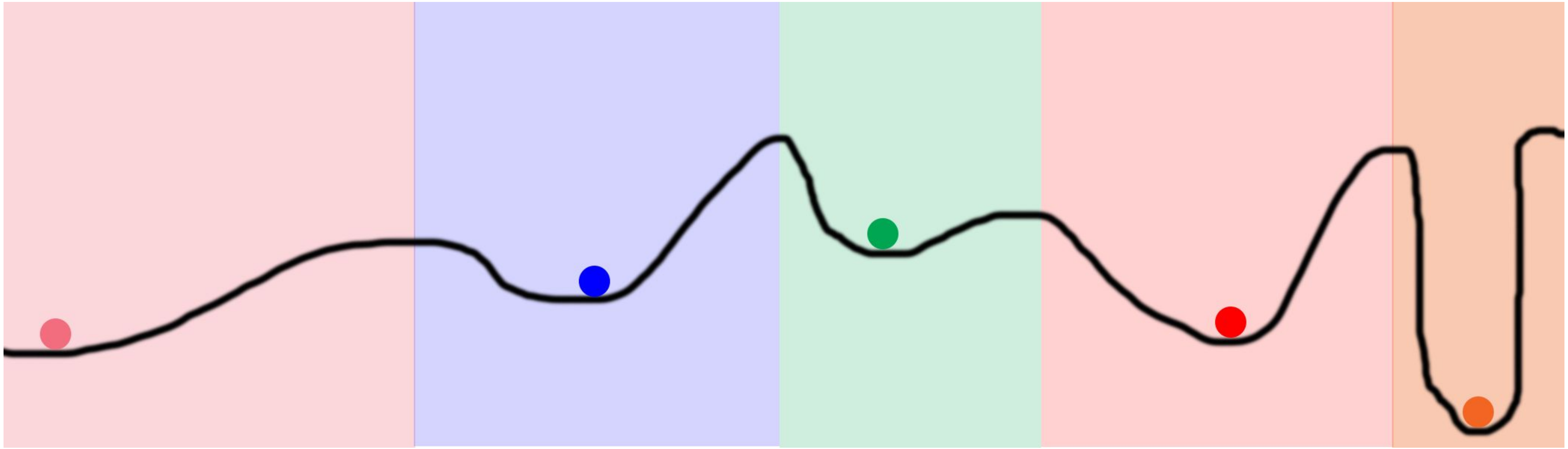
In this diagram, the line represents the error that an ANN produces for its various weights.



Artificial Neural Networks: Types

Dropout (This is pretty cutting edge stuff)

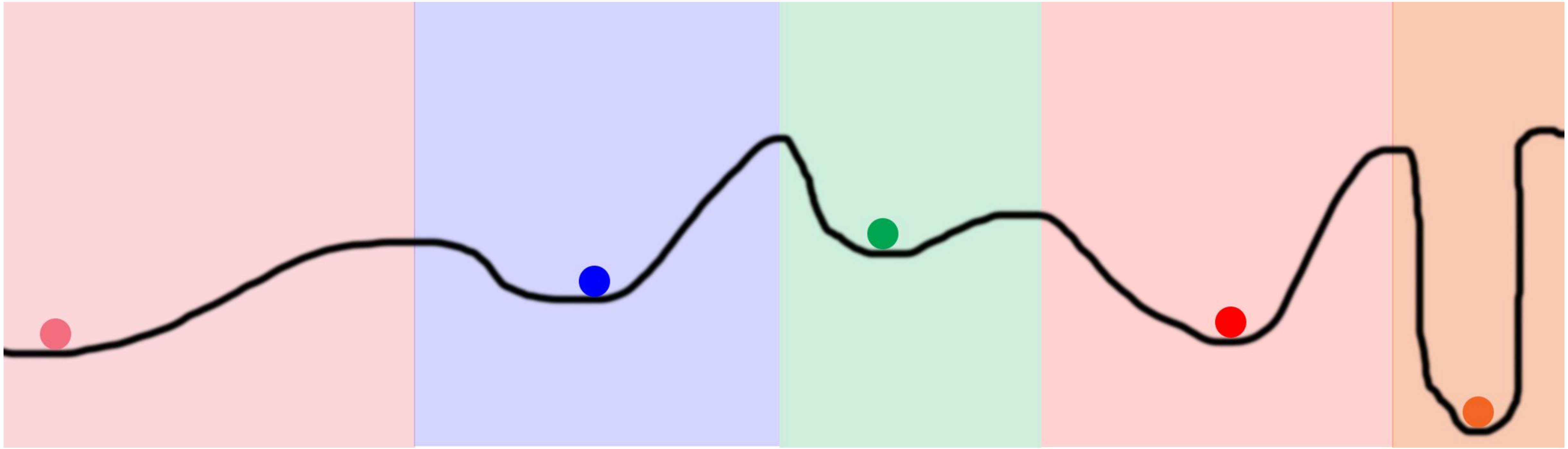
We know that we can use gradient descent to help our weights (the spheres, in this diagram) to reach the minimum points for their specific error boundary.



Artificial Neural Networks: Types

Dropout (This is pretty cutting edge stuff)

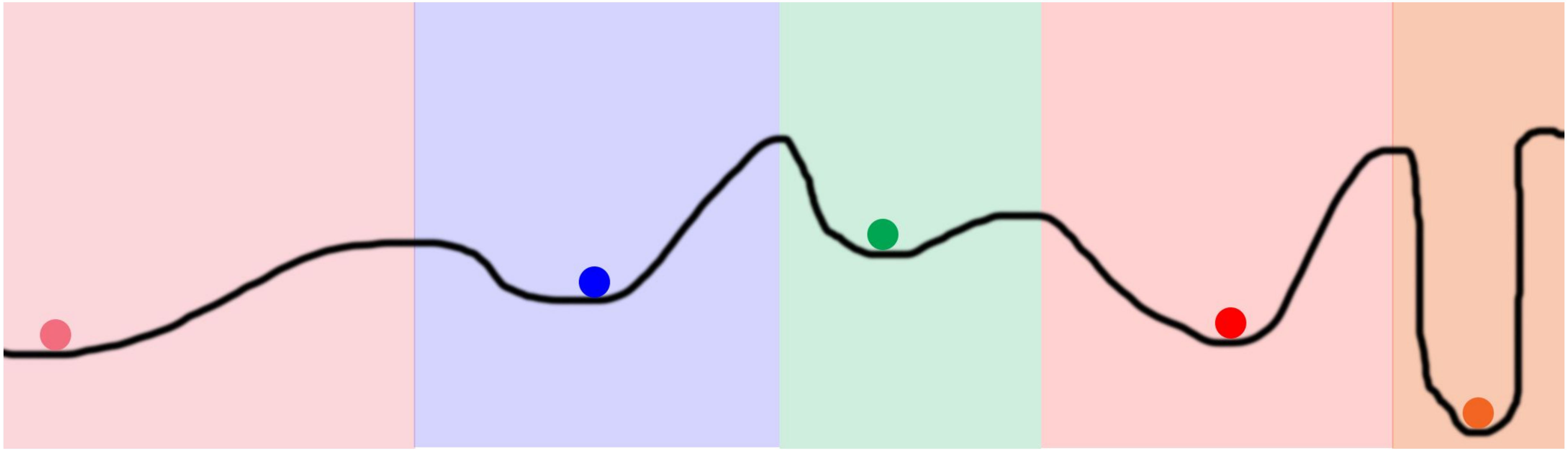
However, because weights are initialized randomly, what happens if two spheres initialize in the same color? This is expensive and redundant, and we'd like to prevent this from happen (It's a pretty frequent problem)



Artificial Neural Networks: Types

Dropout (This is pretty cutting edge stuff)

We prevent this with **dropout.**



Artificial Neural Networks: Types

Dropout (This is pretty cutting edge stuff)

Dropout can prevent two weights from converging to the same local minima by randomly turning nodes off during forward propagation (the normal flow of information through the network structure).

We do this for each neuron in the hidden layer.

It then back-propagates with all the nodes turned on.

Artificial Neural Networks: Types

Dropout (This is pretty cutting edge stuff)

By forcing the ANN to learn multiple independent representations of the same exact data by randomly disabling neurons in the learning phase, the neurons are prevented from co-adapting too much which makes overfitting less likely.

Artificial Neural Networks: Types

Dropout (This is pretty cutting edge stuff)

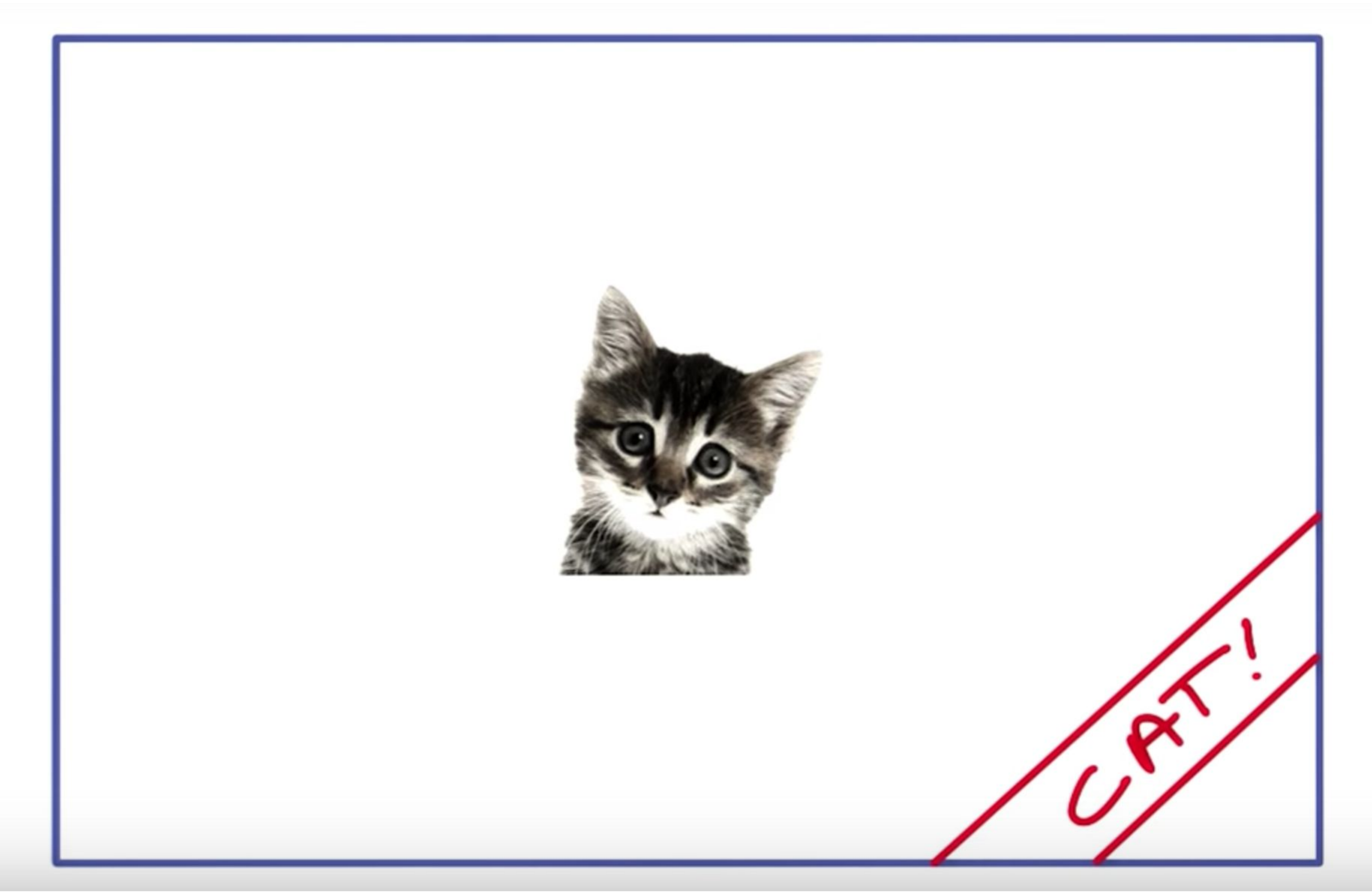
By forcing the ANN to learn multiple independent representations of the same exact data by randomly disabling neurons in the learning phase, the neurons are prevented from co-adapting too much which makes overfitting less likely.

By not co-adapting, neurons in the hidden layer focus on features that are generally more useful to the network as a whole

Part II: Convolutional Neural Networks

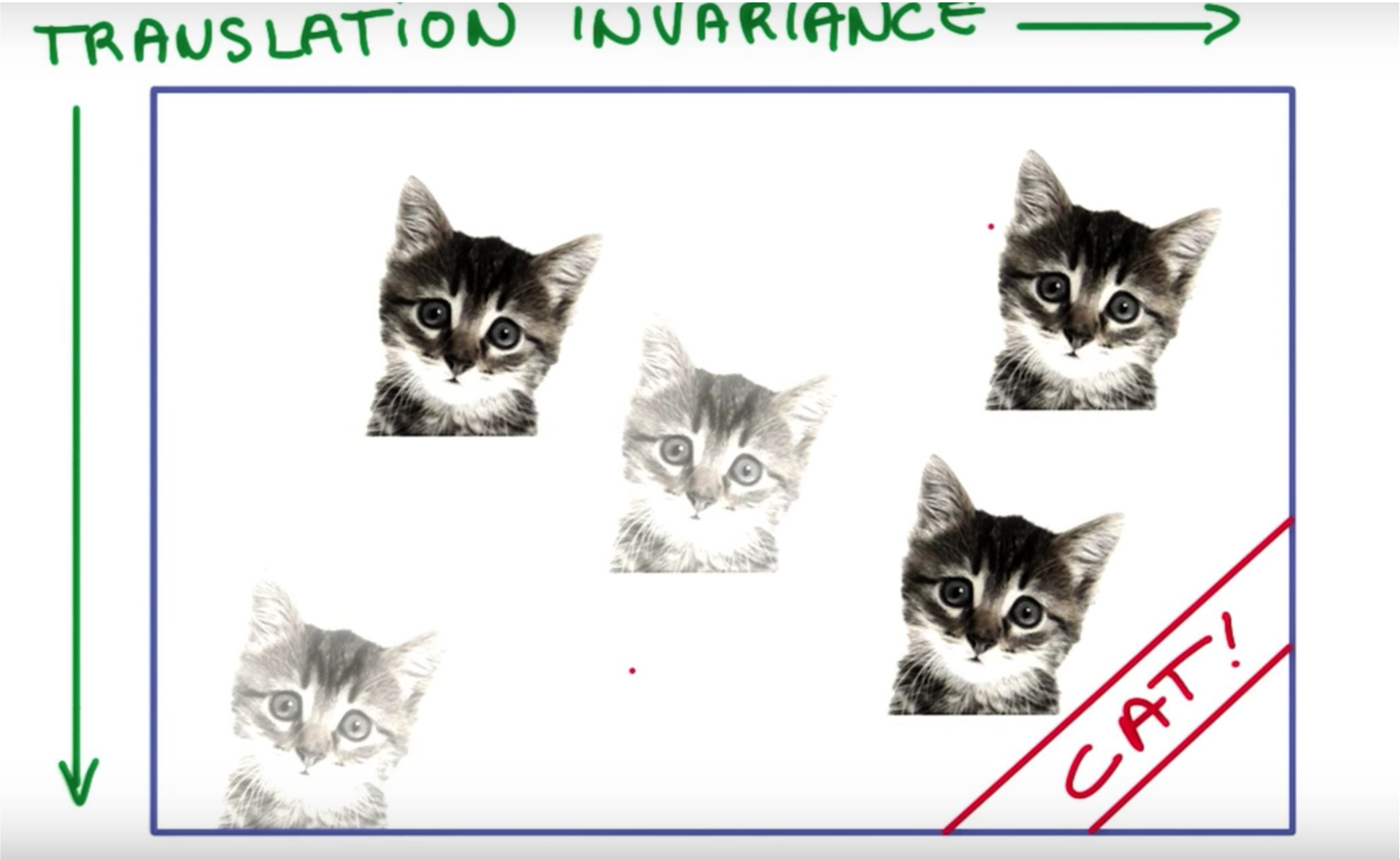
Artificial Neural Networks: ConvNet

The Convolutional Neural Network



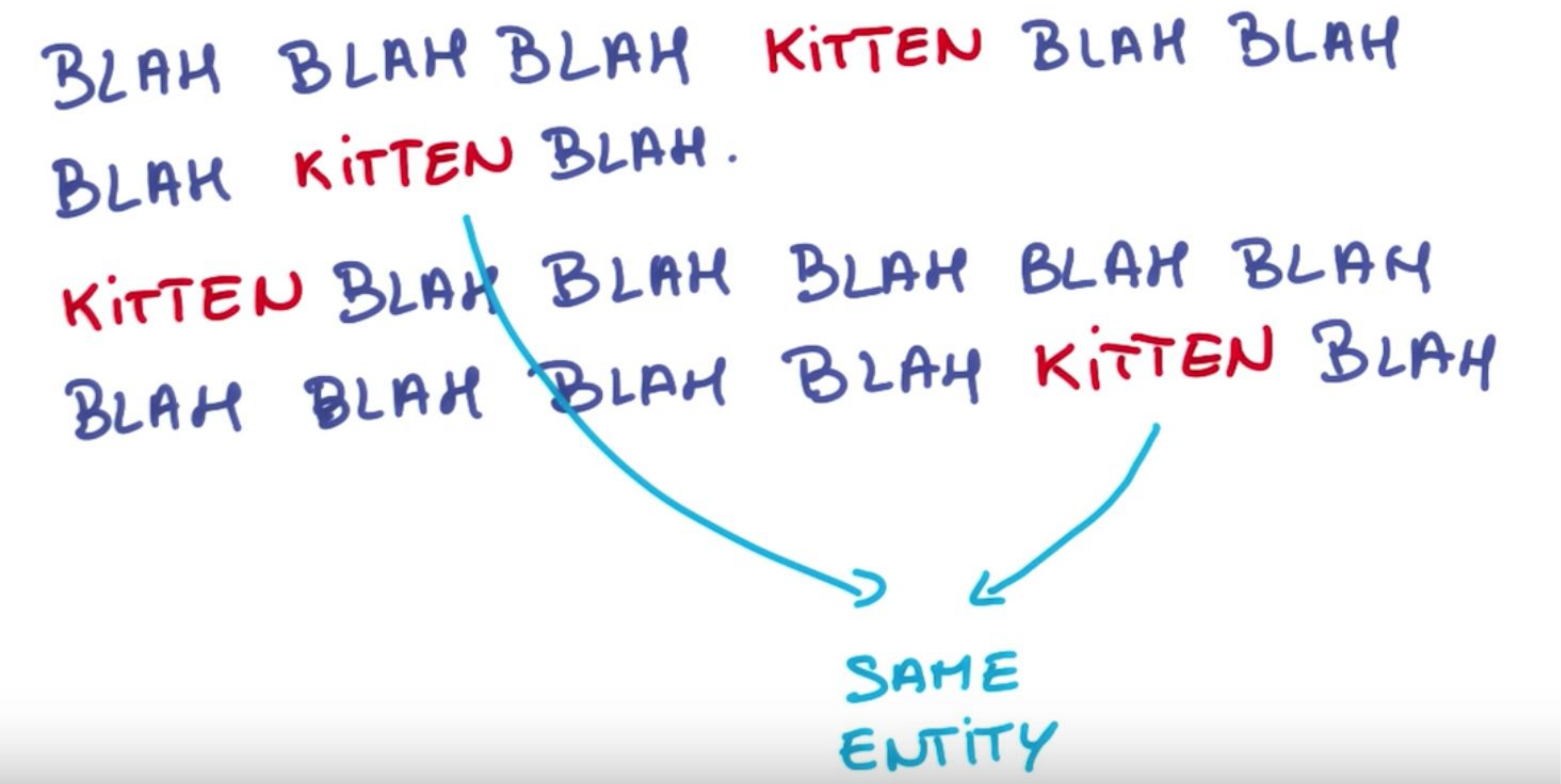
Artificial Neural Networks: ConvNet

The Convolutional Neural Network



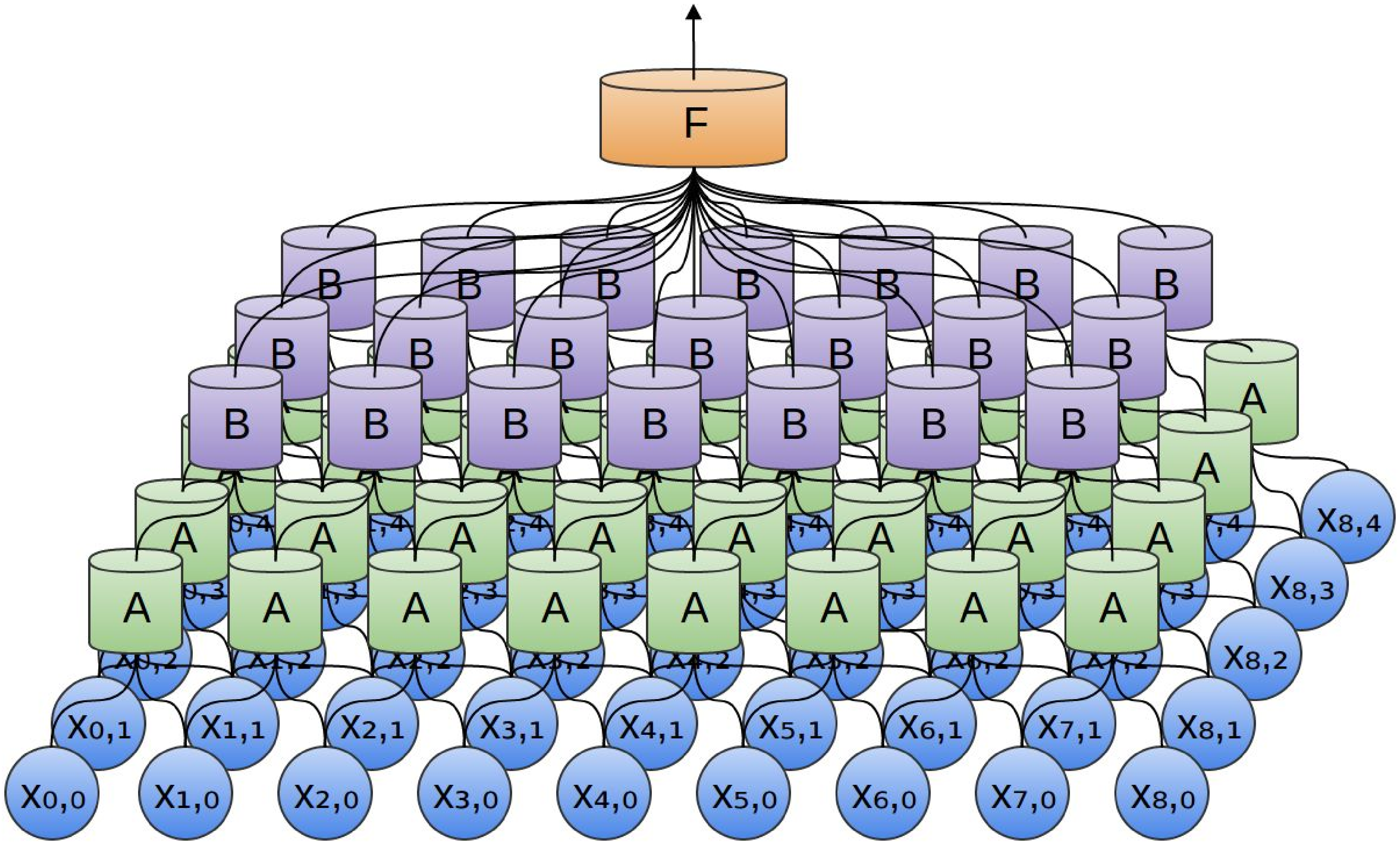
Artificial Neural Networks: ConvNet

The Convolutional Neural Network



Artificial Neural Networks: ConvNet

The Convolutional Neural Network



Artificial Neural Networks: ConvNet

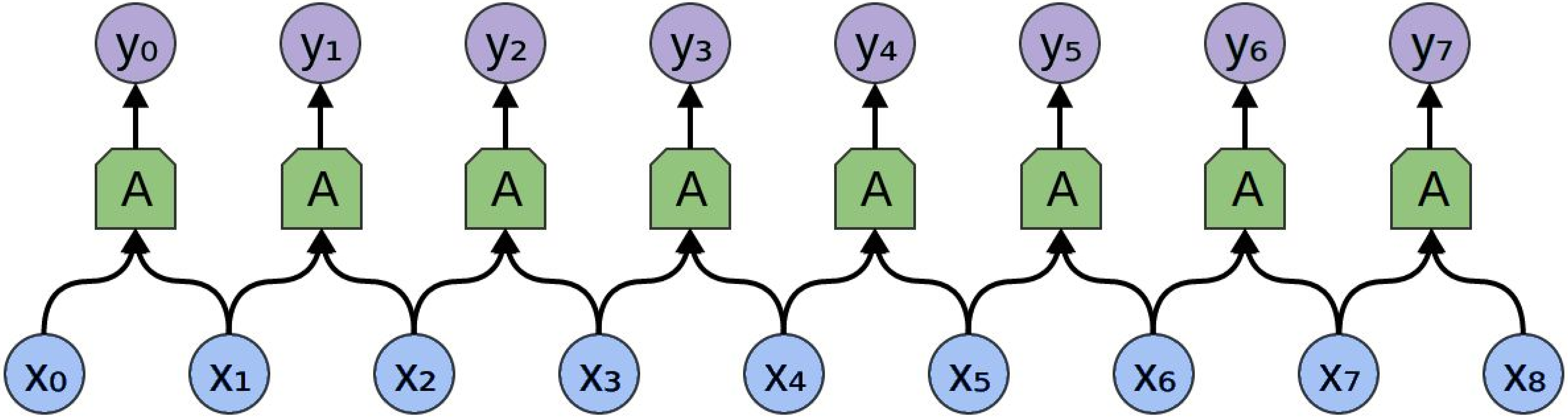
The Convolutional Neural Network

- Convolutional networks use many copies of the same type of neuron
  - This allows the network to be computationally complex with minimal parameters
  - It also allows for faster learning and reduced error

Artificial Neural Networks: ConvNet

The Convolutional Neural Network

- CNN’s work like a *map* function; they apply a function to a small window around every element



Artificial Neural Networks: ConvNet

The Convolutional Neural Network

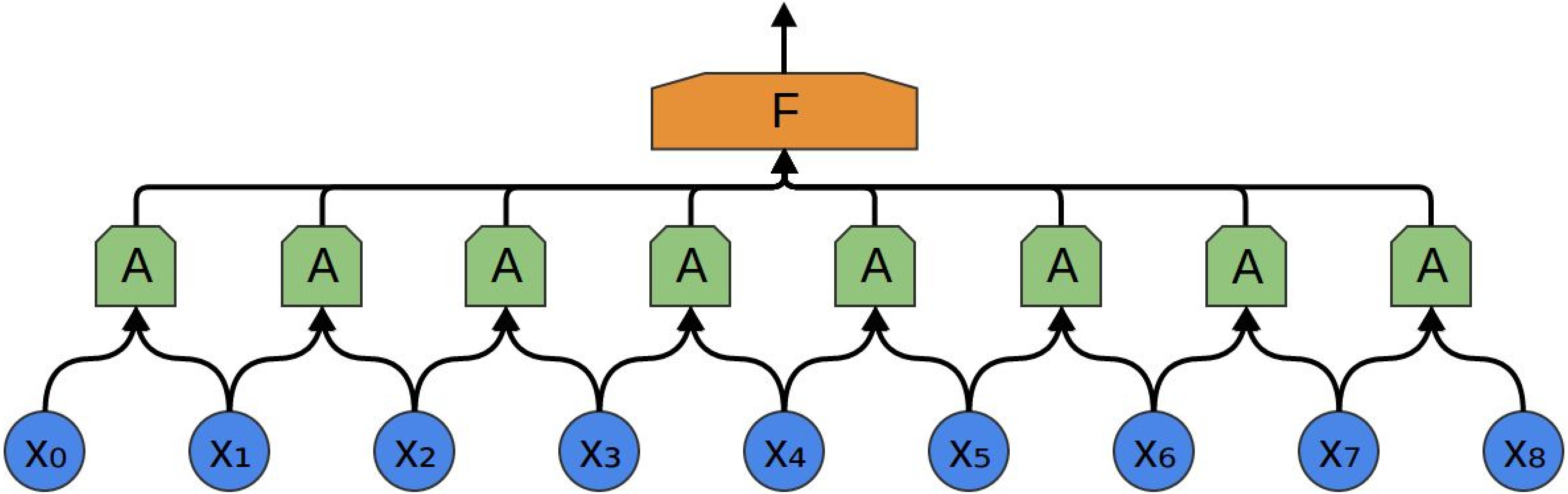
Let’s take a look at how CNNs work:

Imagine that we have a series of input samples **X***0* through **X***8*. These elements will represent our **input layer.**

Artificial Neural Networks: ConvNet

The Convolutional Neural Network

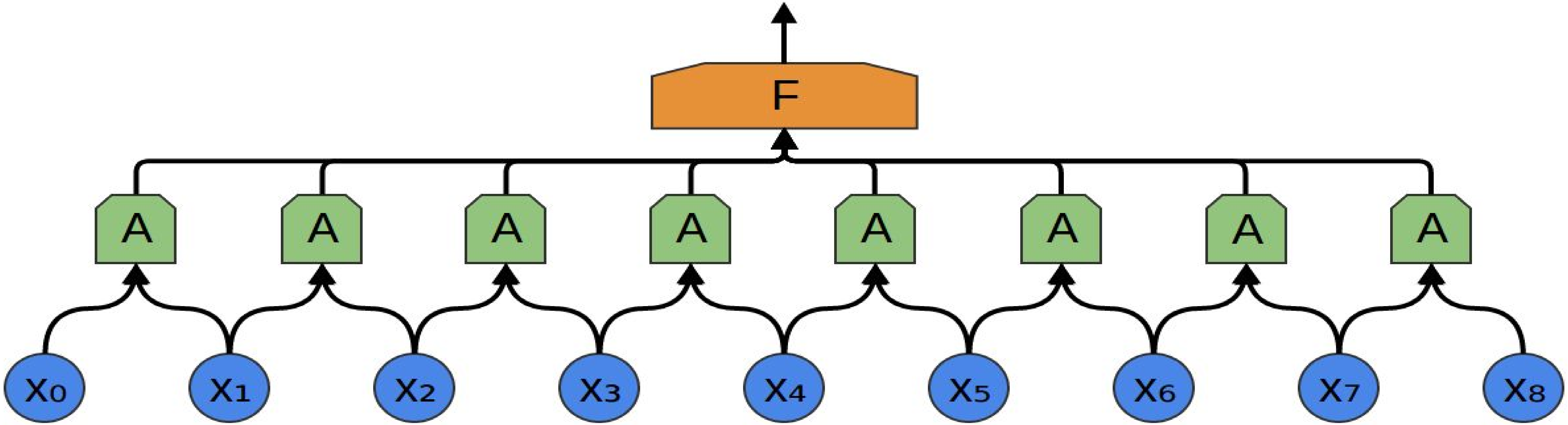
We can begin to construct our network by initializing a set of **artificial neurons**, represented by the “A” nodes here. We call this row of “A” the **convolutional layer**.



Artificial Neural Networks: ConvNet

The Convolutional Neural Network

Here, each neuron ***A*** is looking at a small segment of the data.pon this In this case, the convolutional layer is 1 dimensional. However, we can improve upon this with something called the **window.**



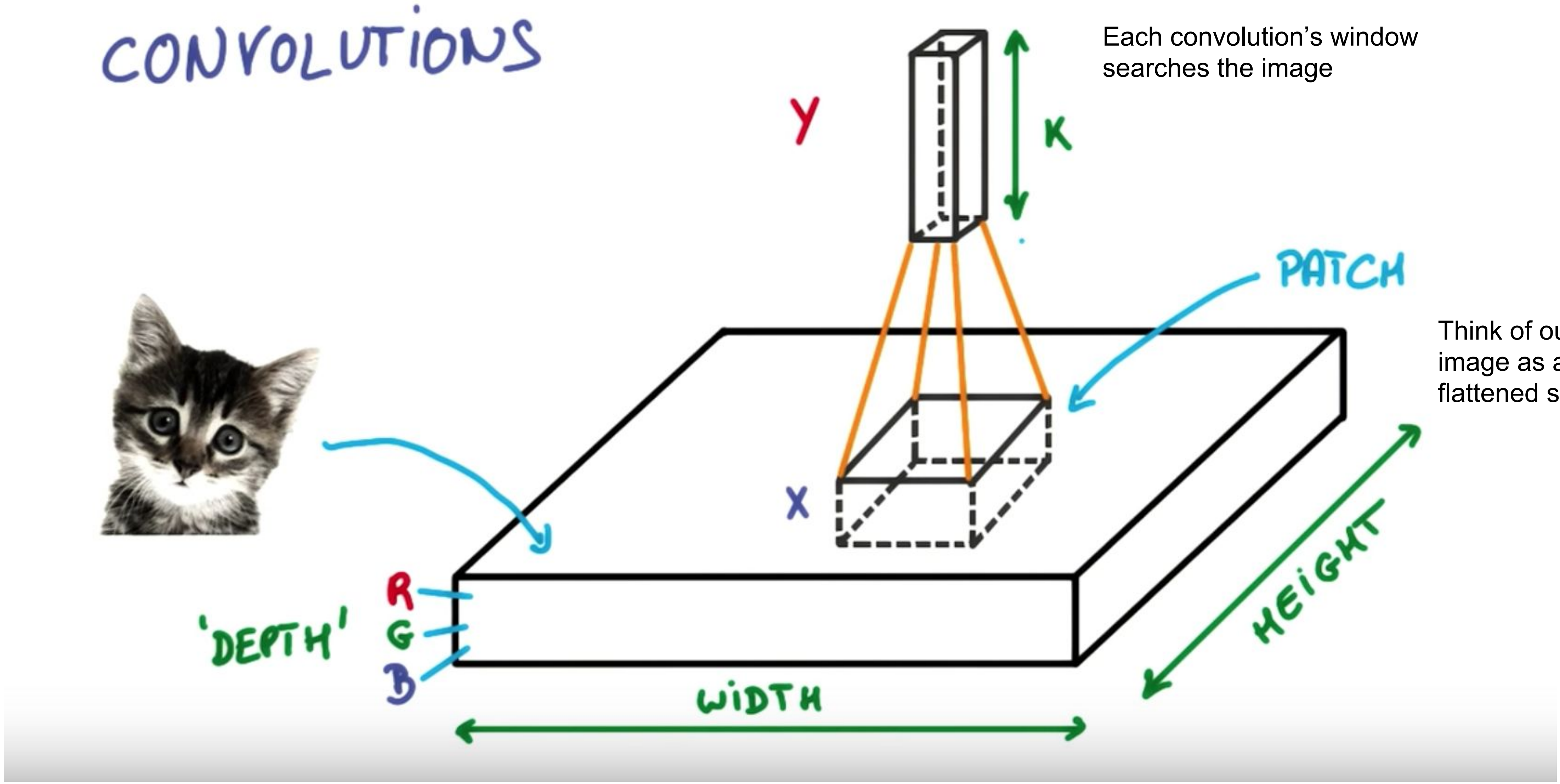
Artificial Neural Networks: ConvNet

The Convolutional Neural Network

Let's think back to our cat example:

Artificial Neural Networks: ConvNet

The Convolutional Neural Network

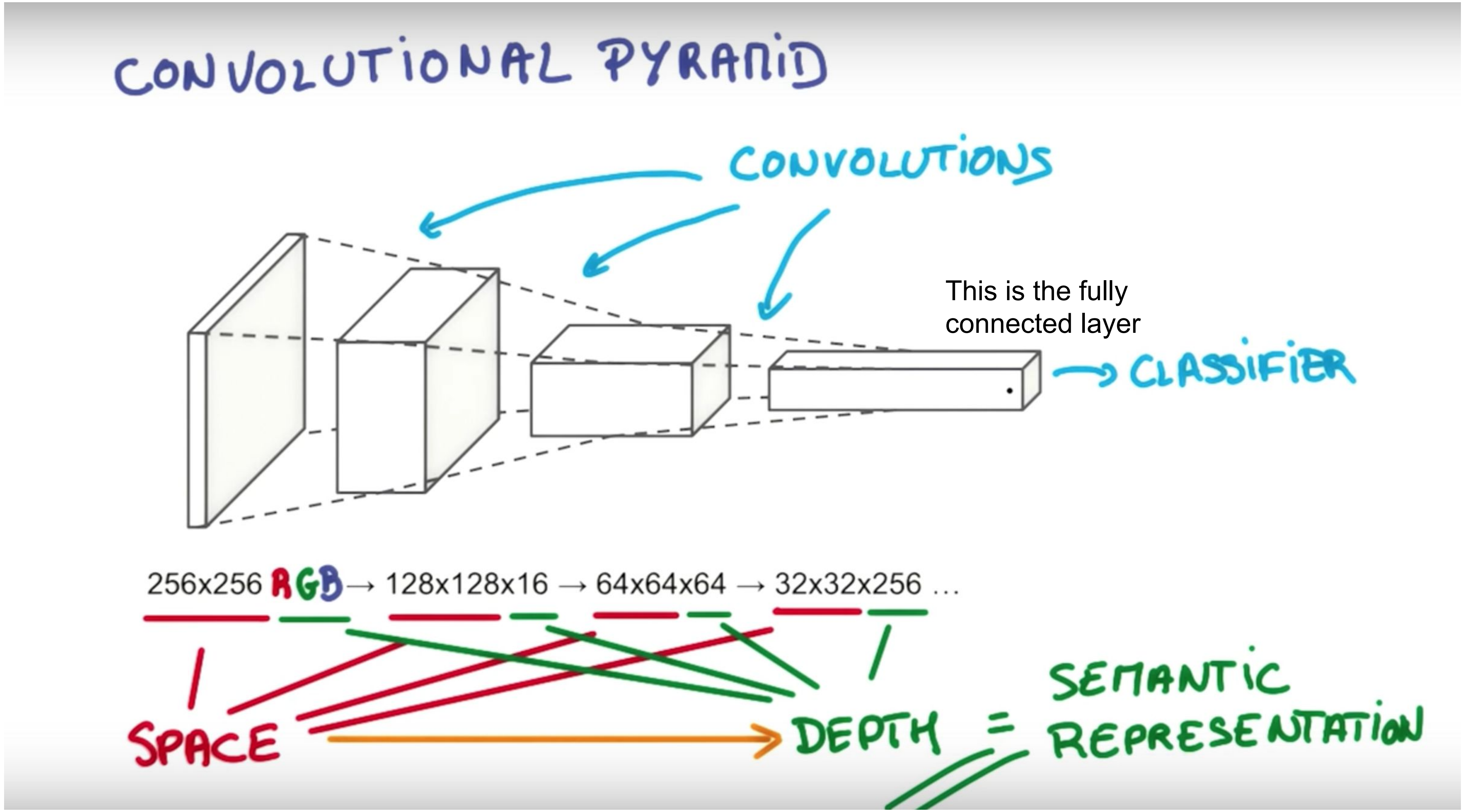


Each convolution's window searches the image

Think of our image as a flattened square

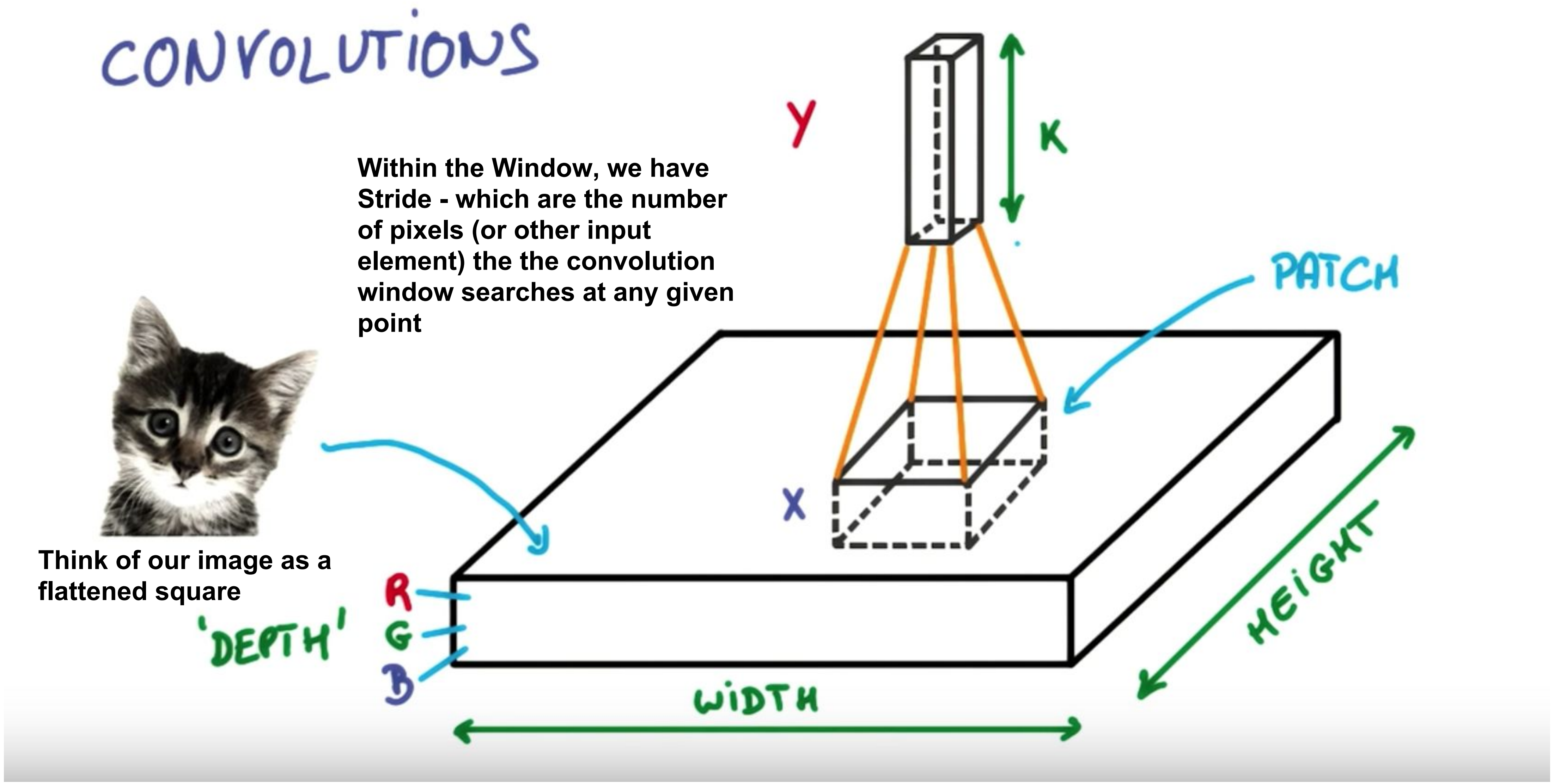
Artificial Neural Networks: ConvNet

The Convolutional Neural Network



Artificial Neural Networks: ConvNet

The Convolutional Neural Network



Within the Window, we have Stride - which are the number of pixels (or other input element) the the convolution window searches at any given point

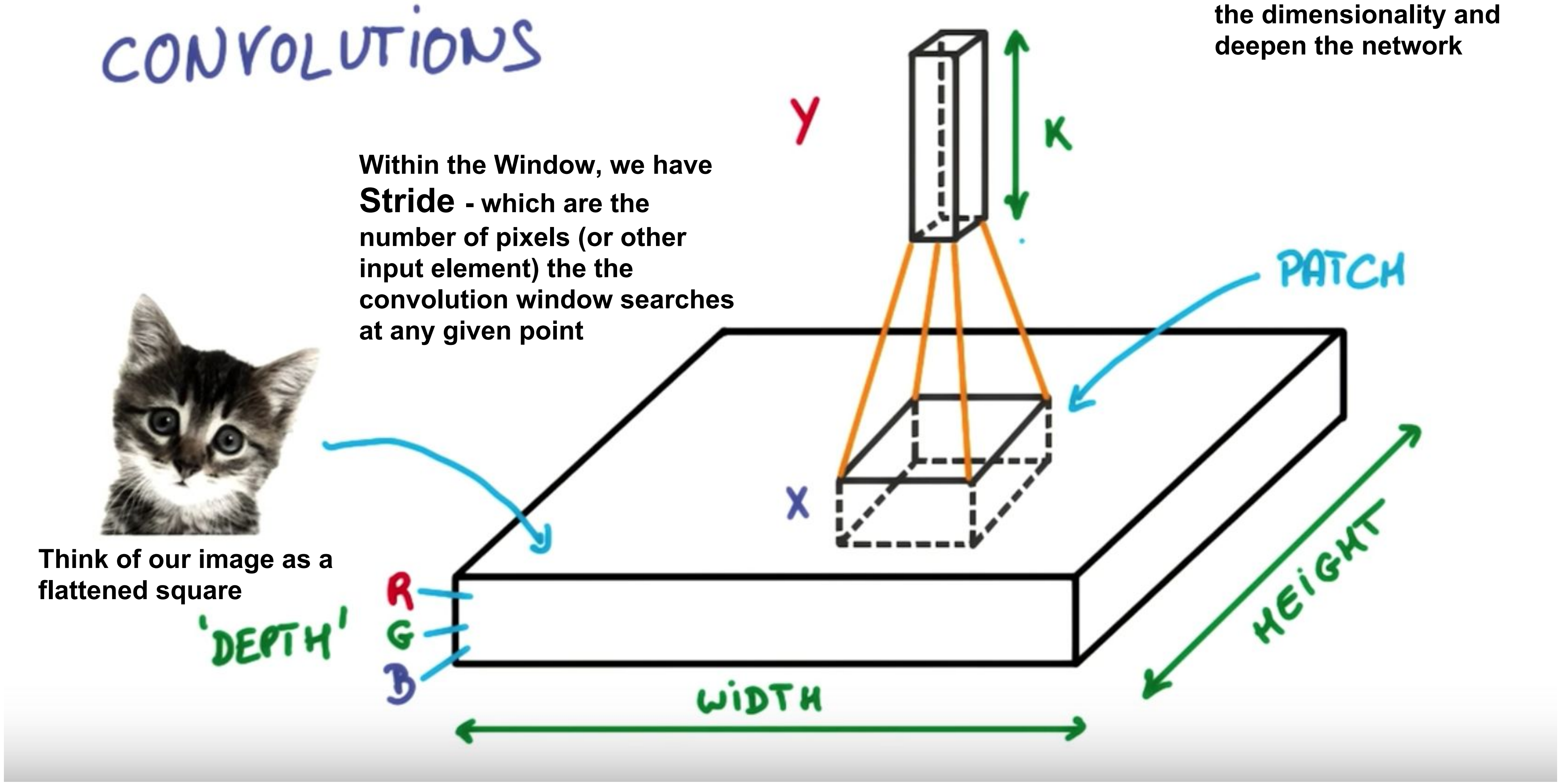
Think of our image as a flattened square

Artificial Neural Networks: ConvNet

The Convolutional Neural Network

Stride helps us reduce the dimensionality and deepen the network

Within the Window, we have **Stride** - which are the number of pixels (or other input element) the the convolution window searches at any given point

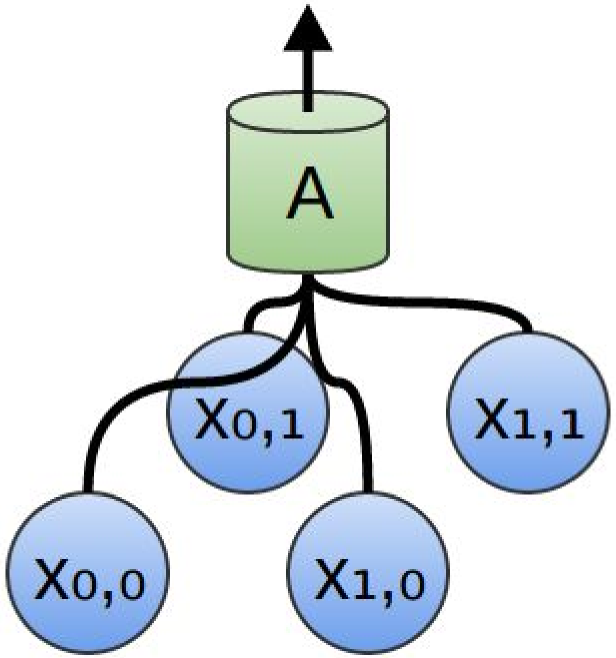


Think of our image as a flattened square

Artificial Neural Networks: ConvNet

The Convolutional Neural Network

In the context of our toy network here, the window for each convolution would look something like the figure on the right.

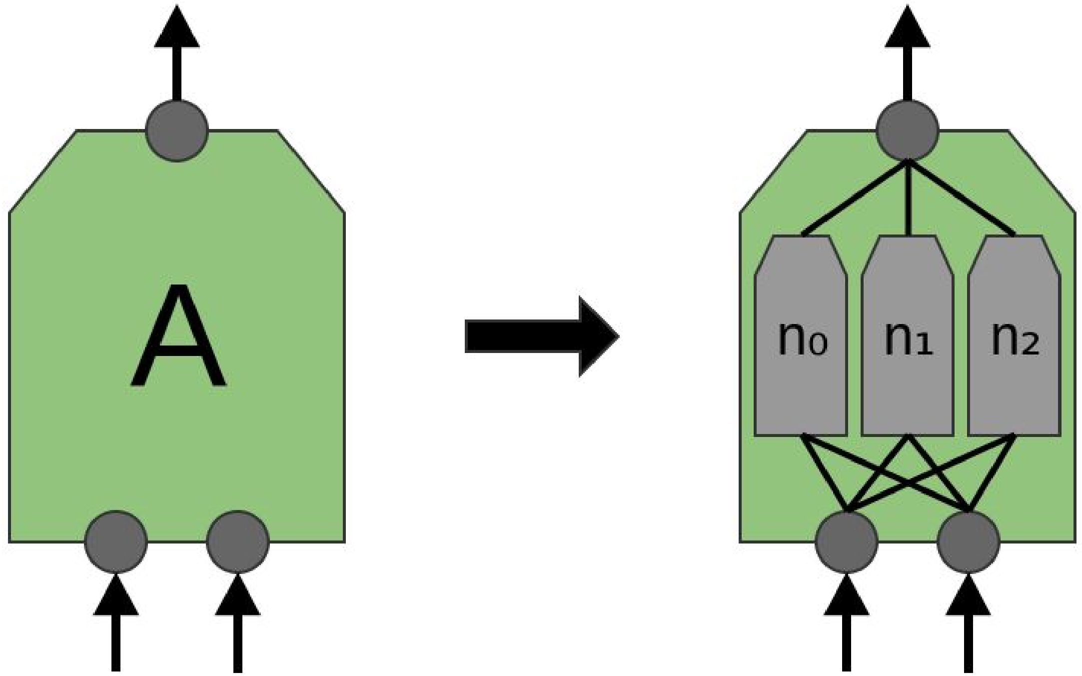


Convolutional networks that use a window, instead of looking at one specific point at a time are called **2D Convolutional Networks**.

Artificial Neural Networks: ConvNet

The Convolutional Neural Network

Each of these 2D convolutions will look at something different, perhaps edges, perhaps color. Each will focus on a specific attribute

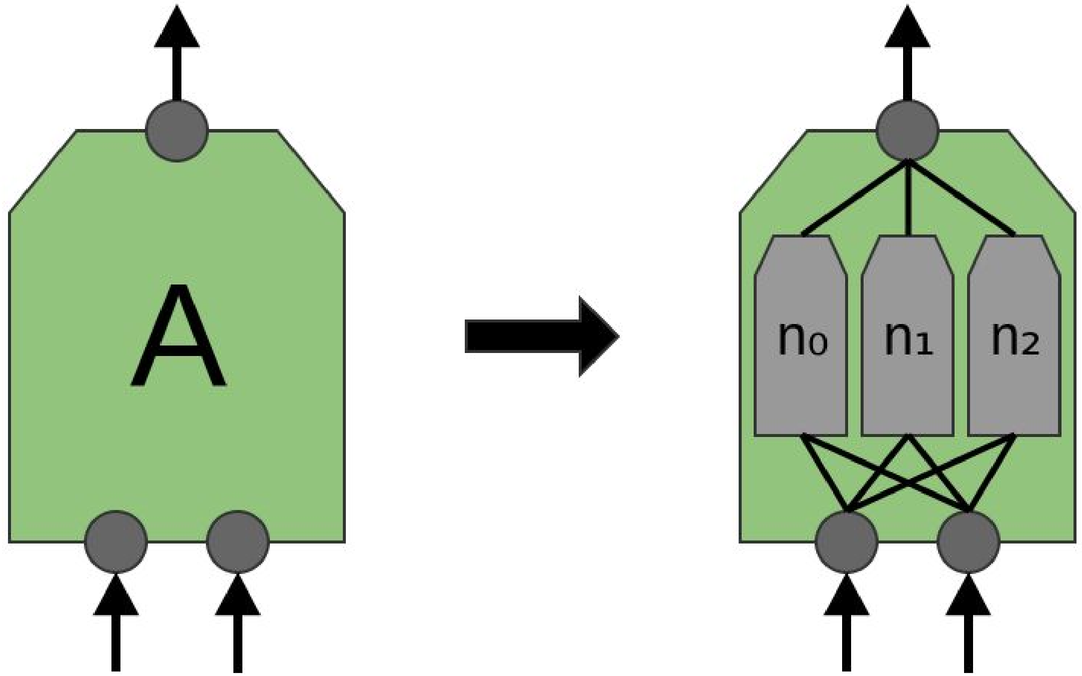


Artificial Neural Networks: ConvNet

The Convolutional Neural Network

Each of these 2D convolutions will look at something different, perhaps edges, perhaps color. Each will focus on a specific attribute

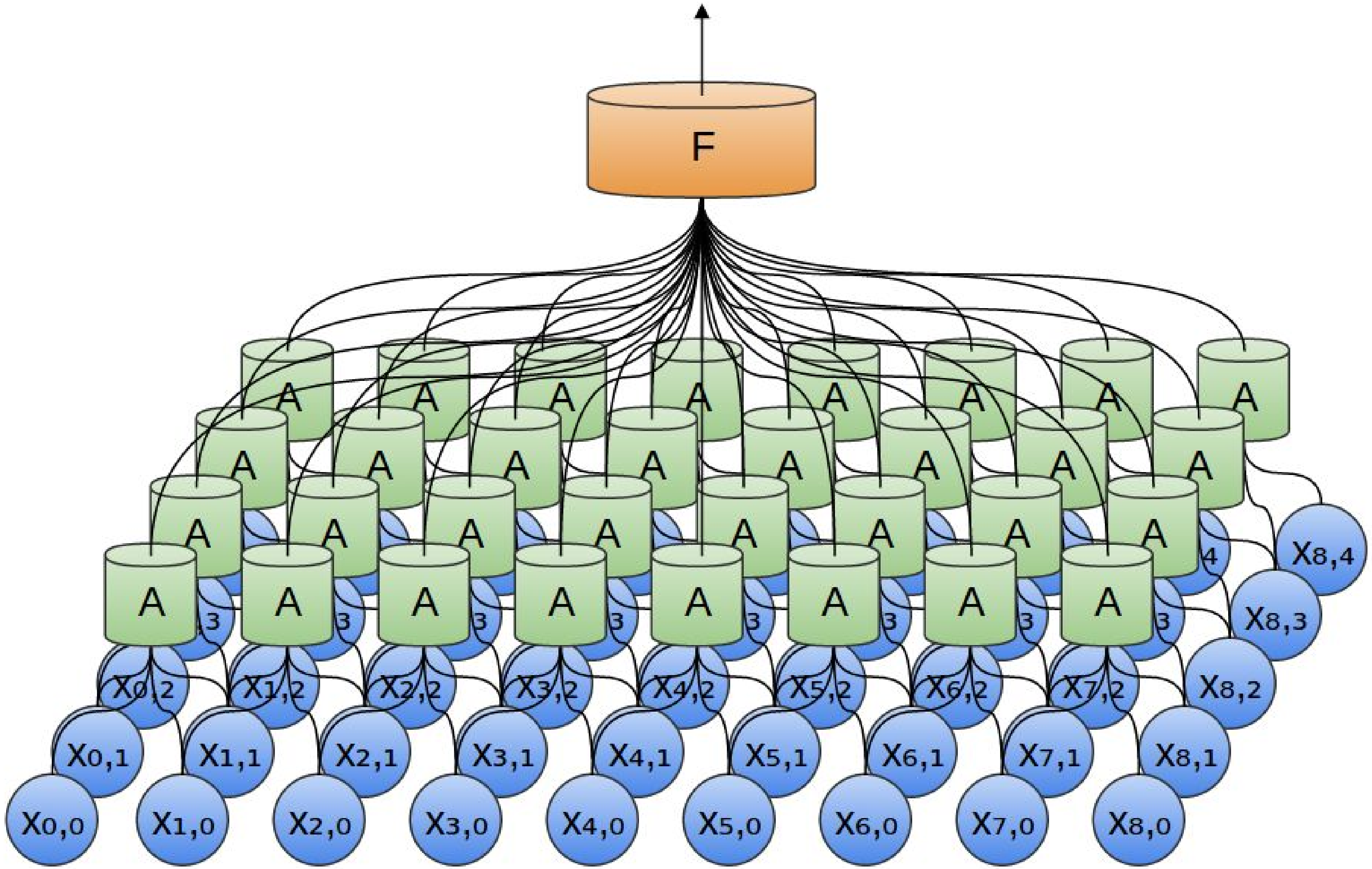
In this way, we have (in a sense), a network within a network.



Artificial Neural Networks: ConvNet

The Convolutional Neural Network

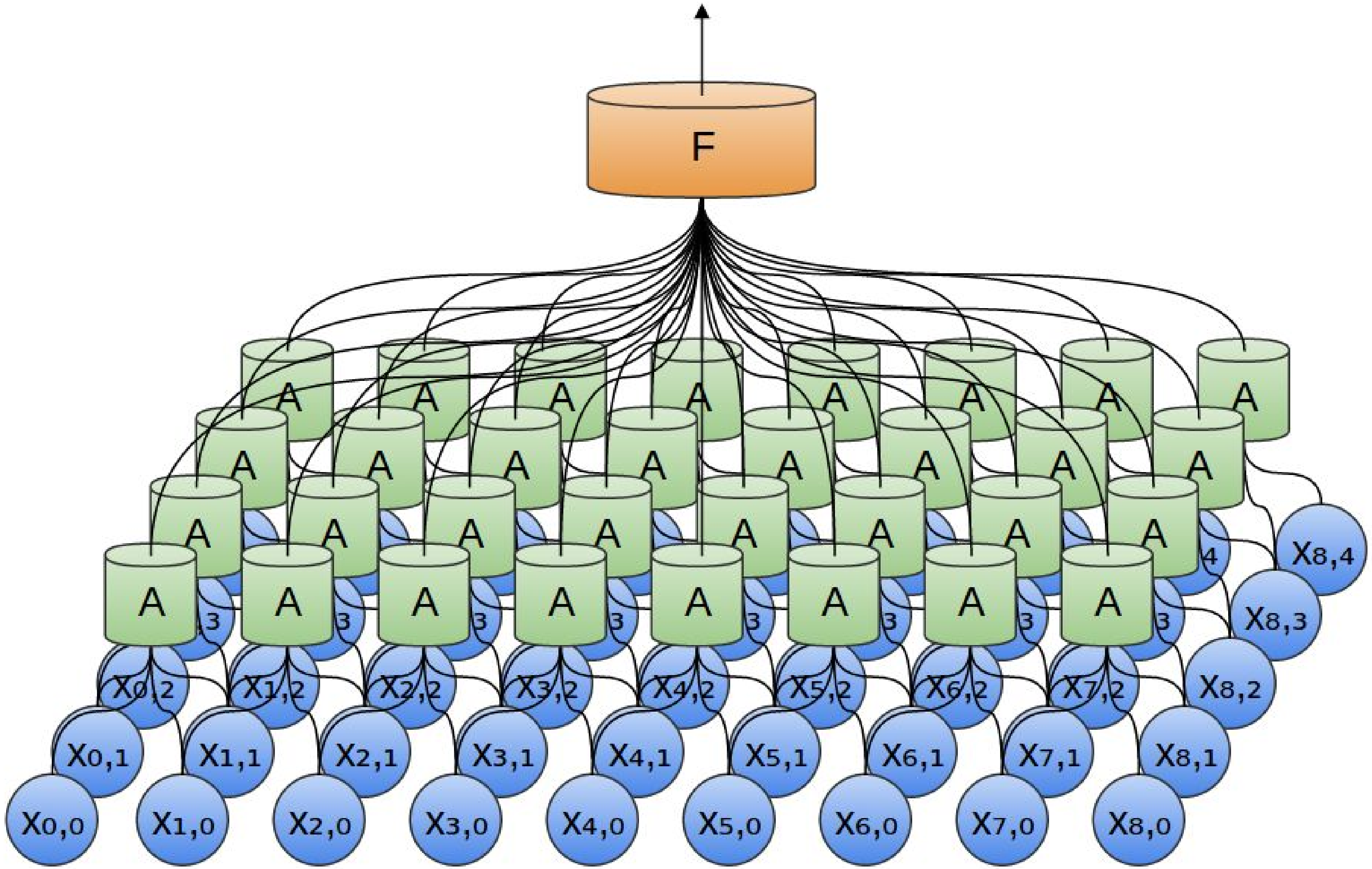
Each of these neurons will perform some form of computation. Together, they pass the results of their computations onto the **fully connected layer (**represented here by **F).**



Artificial Neural Networks: ConvNet

The Convolutional Neural Network

The forward pass of a fully-connected layer corresponds to one matrix multiplication followed by a bias offset and an activation function.

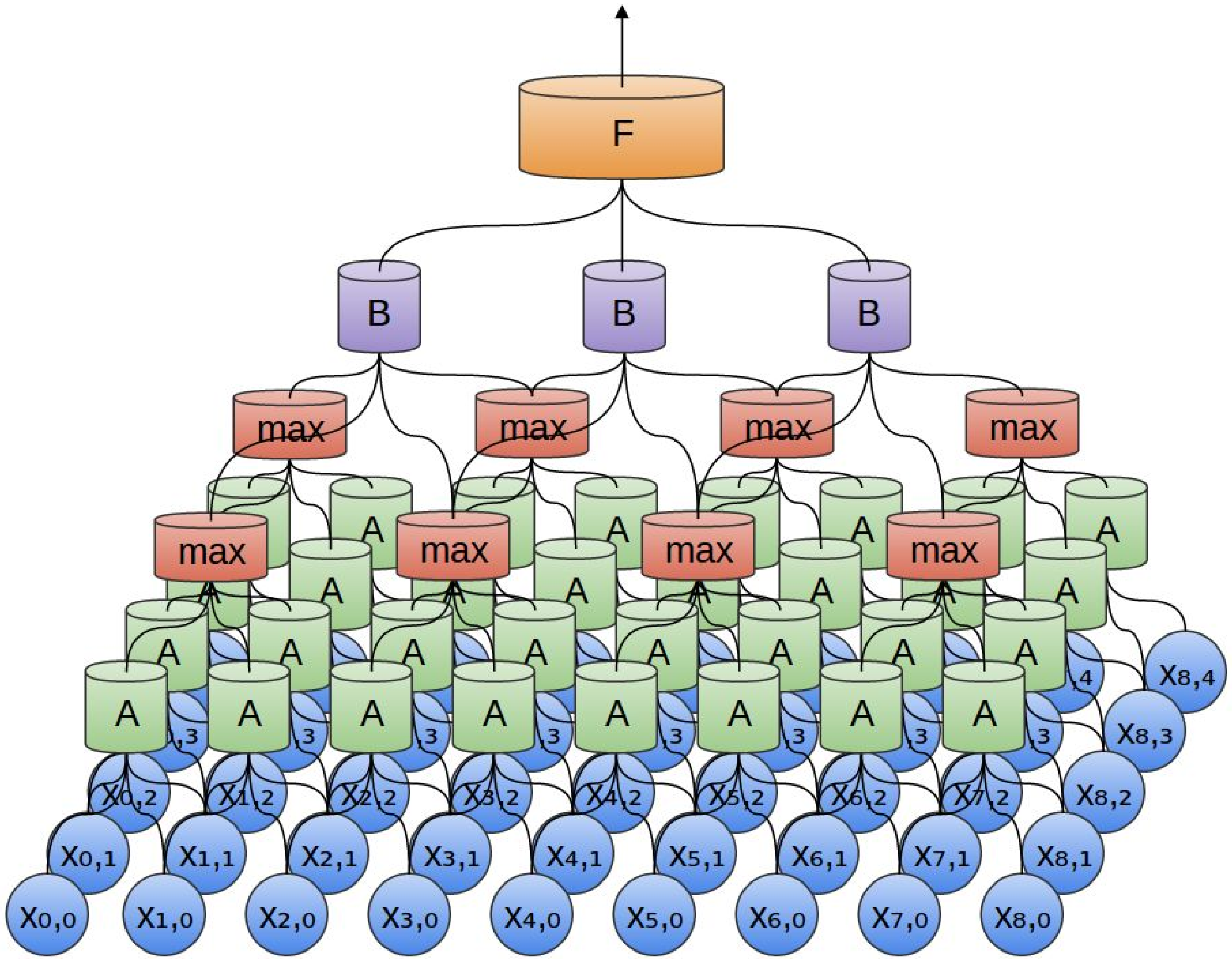


Artificial Neural Networks: ConvNet

The Convolutional Neural Network

Taking this further, convolutional layers can be stacked. Here, we've added a second convolutional layer, **B.**

We've also added something else here, the "max" layers, which is a process called **max-pooling**



Artificial Neural Networks: ConvNet

Pooling

- With pooling; we combine all of the all of the convolutions in a neighborhood
- The most common form of this is **Max Pooling**

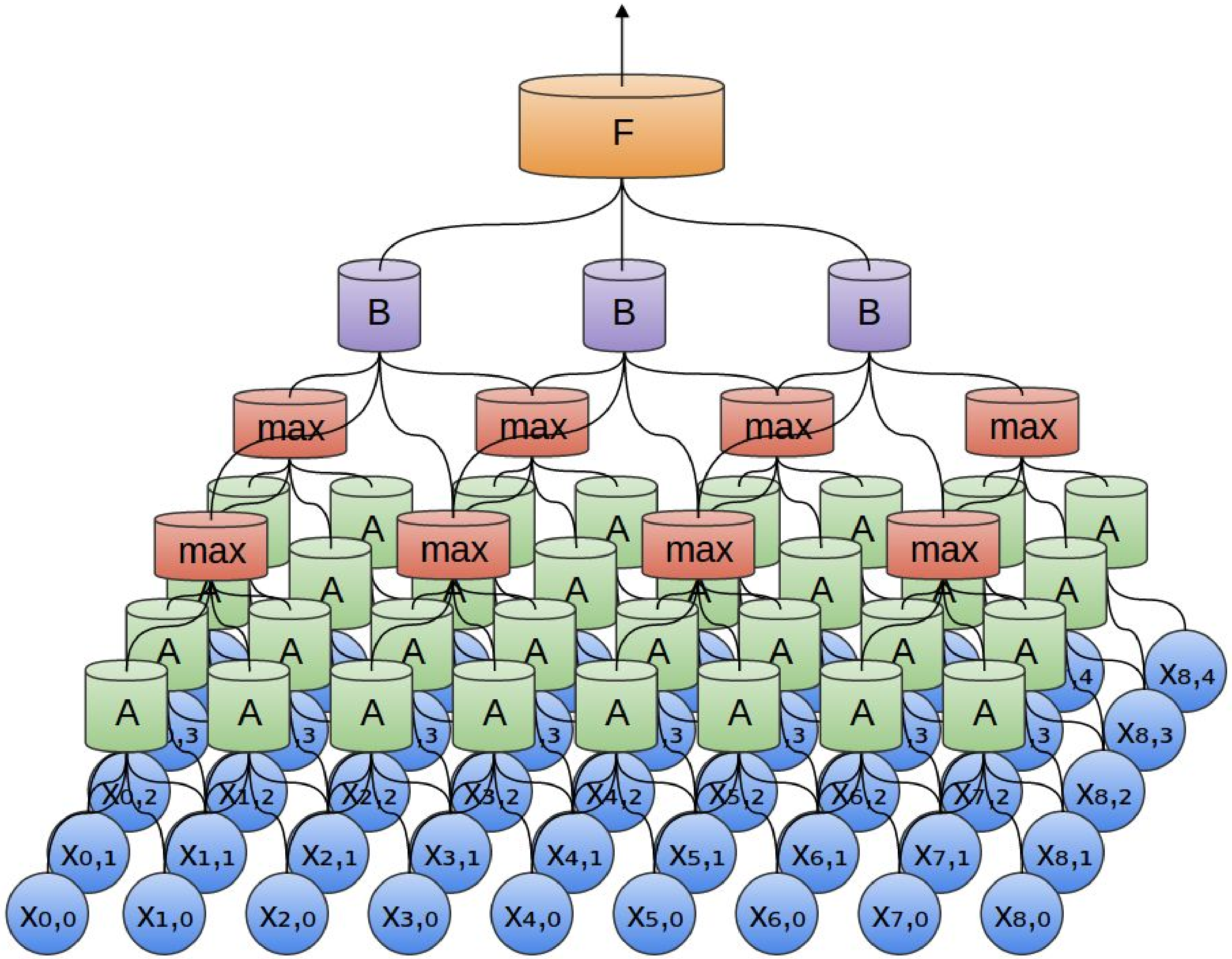
With Max Pooling:

- We look at a small point in the window and compute the maximum for a neighborhood of points around that point
- Max pooling has the benefit of being parameter free
  - However, youdo have to worry about pooling size!
- Max Pooling also increases the computational expense of your network

Artificial Neural Networks: ConvNet

The Convolutional Neural Network

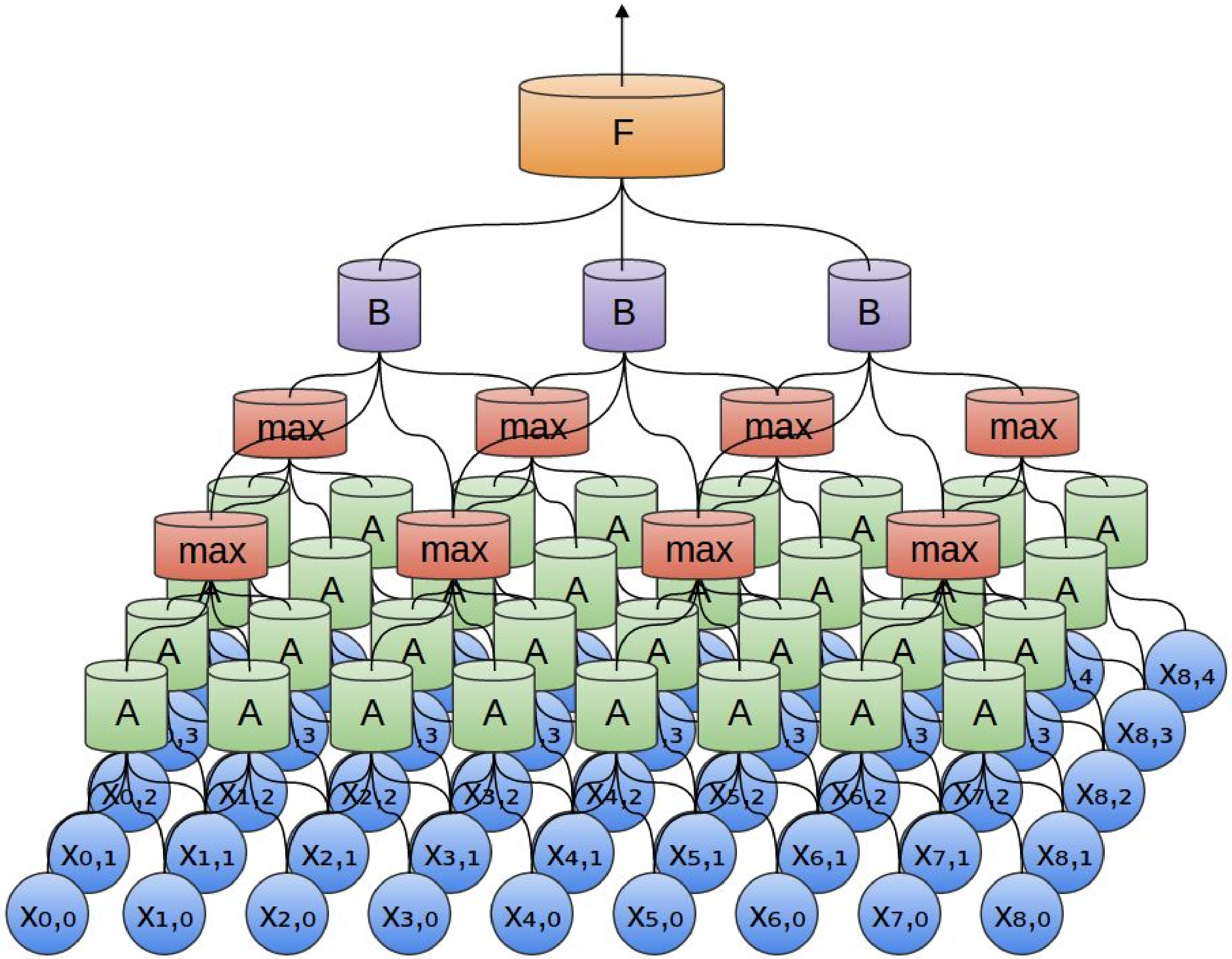
Max-pooling takes samples from the previous convolutional layer, sub-samples them, and returns a single value, thereby decreasing the computational ask on the next convolutional layer



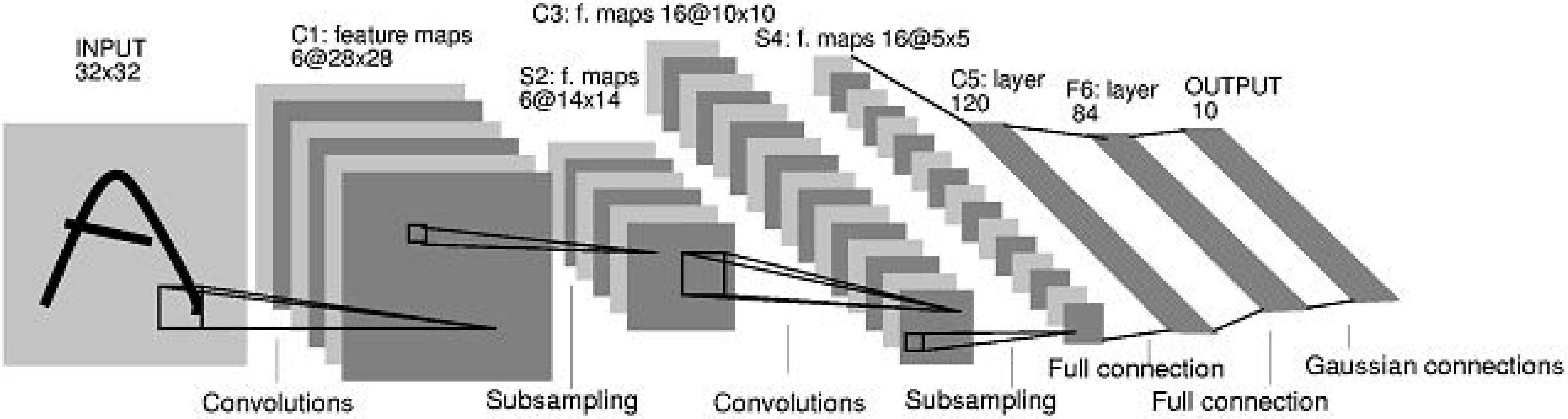
Artificial Neural Networks: ConvNet

The Convolutional Neural Network

What this really boils down to is that, when considering an entire image, we don't care about the exact position of an edge, down to a pixel. It's enough to know where it is to within a few pixels.

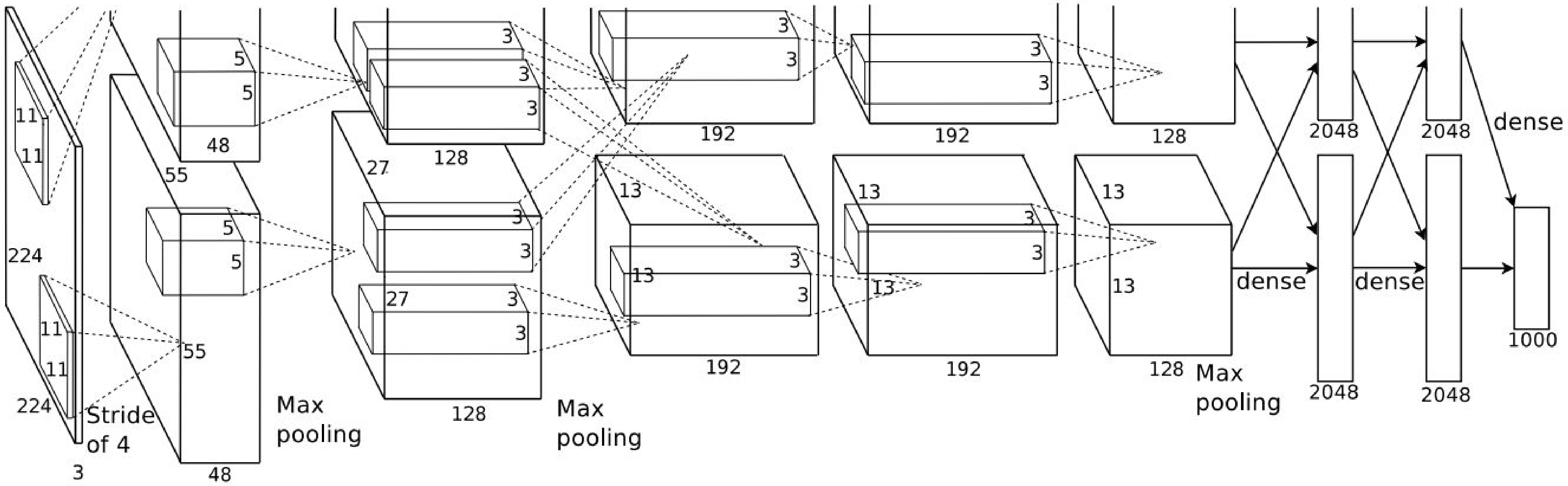


Artificial Neural Networks: ConvNet



Yann LeCun, 1998

Artificial Neural Networks: ConvNet



Krizhevsky, et al. 2013

Artificial Neural Networks: The Output Layer

The Softmax Function

- The softmax function is a generalized version of the logistic function
- When we talk about softmax in deep learning, we are talking about ***softmax regression*** (generalized, multinomial logistic regression)

$$h_\theta(x^{(i)}) = \begin{bmatrix} p(y^{(i)} = 1 | x^{(i)}; \theta) \\ p(y^{(i)} = 2 | x^{(i)}; \theta) \\ \vdots \\ p(y^{(i)} = k | x^{(i)}; \theta) \end{bmatrix} = \frac{1}{\sum_{j=1}^{k} e^{\theta_j^T x^{(i)}}} \begin{bmatrix} e^{\theta_1^T x^{(i)}} \\ e^{\theta_2^T x^{(i)}} \\ \vdots \\ e^{\theta_k^T x^{(i)}} \end{bmatrix}$$

Equation for
Softmax Regression

- We use the softmax function as an output layer when we have a ***multi-class classification problem*** where the classes are ***mutually exclusive.***

Implementing a CNN in Keras

Part III: Recurrent Neural Networks