neural networks

outline

1. rules for neural networks
2. basic introduction
3. activation functions
4. use cases
5. frameworks
6. what next?

rules of neural networks

rules of neural networks

rule #1

do not use neural networks

rules of neural networks

rule #2

neural networks are not magic

rules of neural networks

rule #3

do not use neural networks

rules of neural networks

rule #3

do not use neural networks

unless you have to

rules of neural networks

rule #3

do not use neural networks

unless you have to

or really want to

rules of this lecture

- i am simplifying things
- i hope this is a comprehensible basic introduction to neural networks
- if you want more this should serve as a jumping off point to future work/study

rules of this lecture

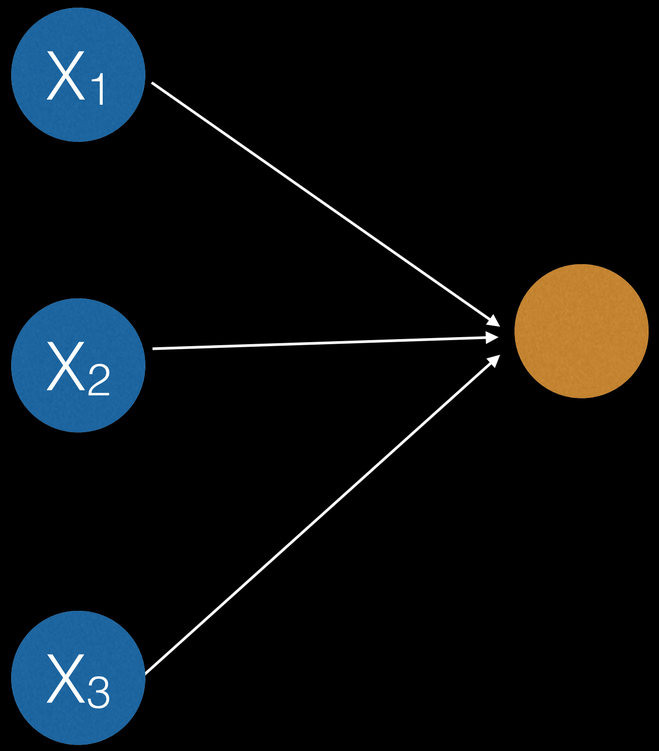
- i am simplifying things
- i hope this is a comprehensible basic introduction to neural networks
- if you want more this should serve as a jumping off point to future work/study
- but you shouldn’t use neural nets

what is a neural network

- series of nodes
- connected with weights
- non-linear activation functions
- can recreate other ml algorithms
  - linear regression
  - logistic regression

single layer neural network

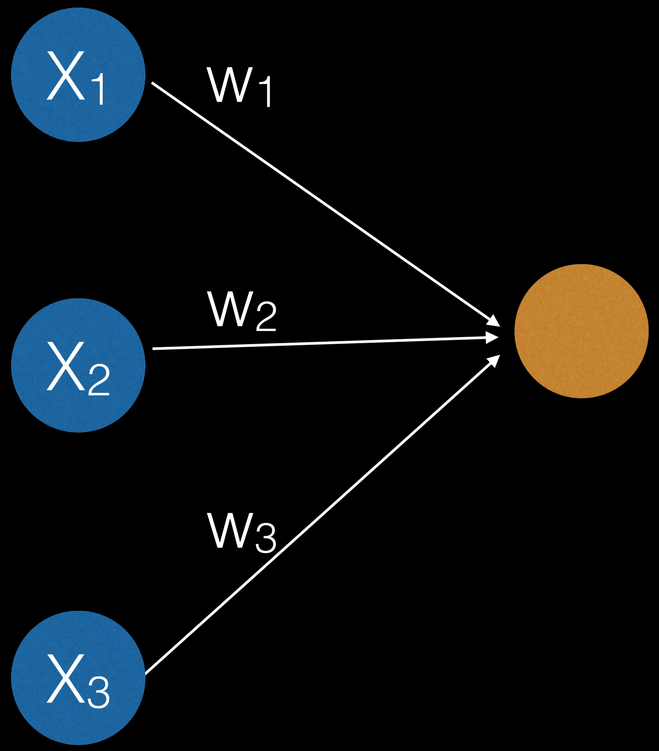
single layer neural network



- X_n are data

Input layer Output layer

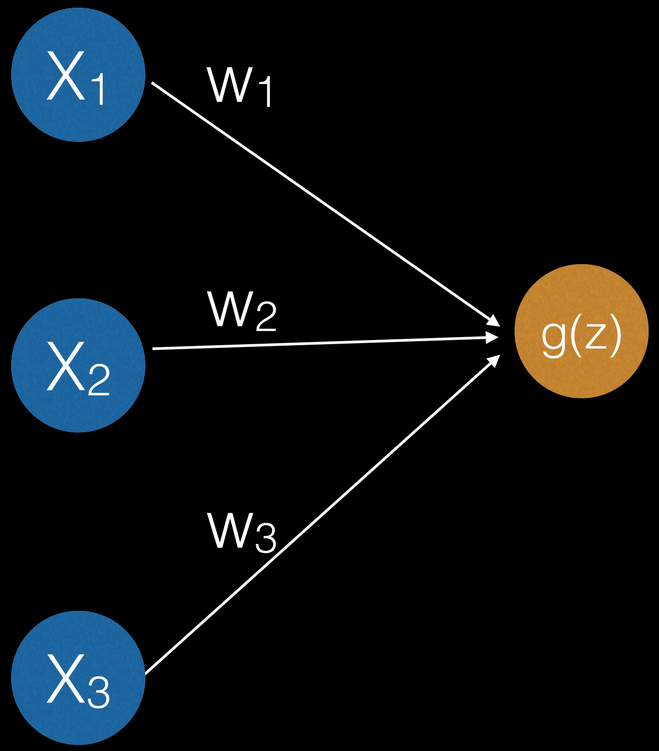
single layer neural network



- X_n are data
- w_n are the weights

Input layer Output layer

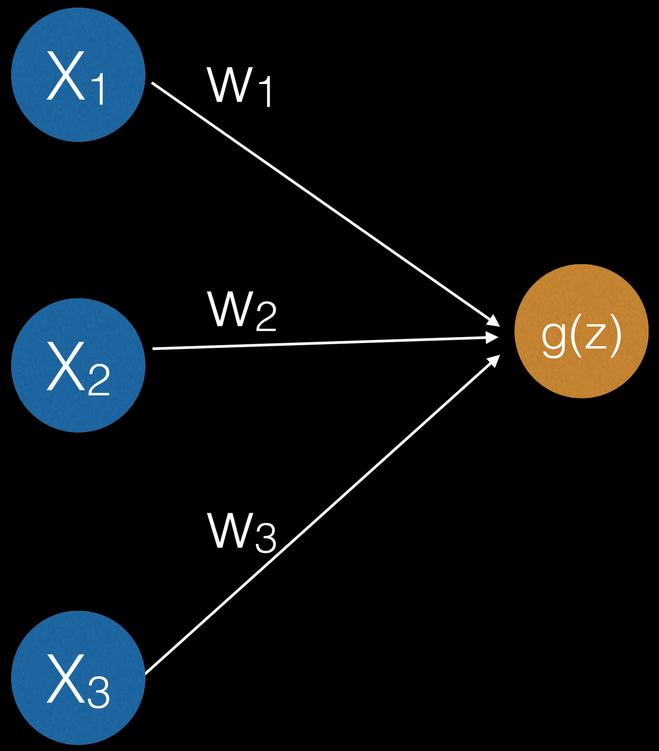
single layer neural network



- X_n are data
- w_n are the weights
- z is weighted input. z = WX + b, b is the bias
- g(z) is the activation (more on those later)

Input layer Output layer

single layer neural network

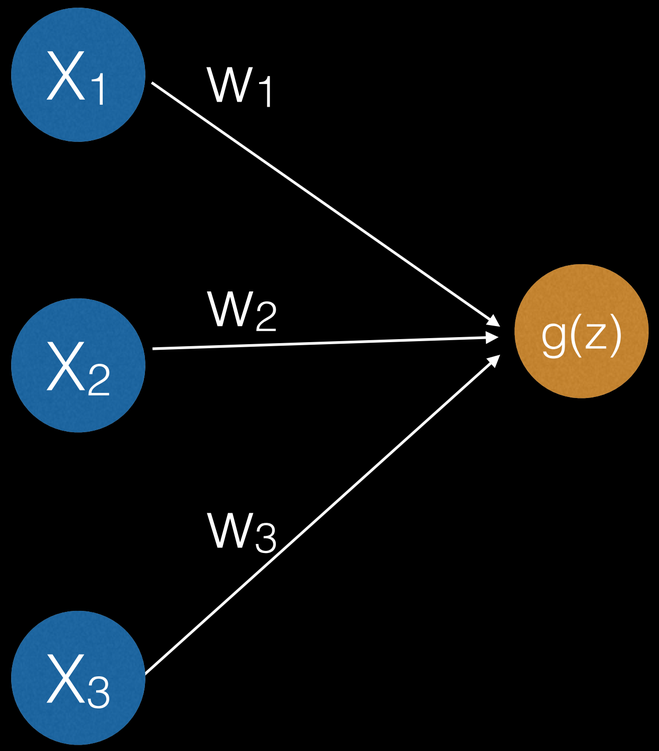


- X_n are data
- w_n are the weights
- z is weighted input. z = WX + b, b is the bias
- g(z) is the activation (more on those later)
- weights and biases can be found with backpropagation

Input layer Output layer

single layer neural network

linear regression

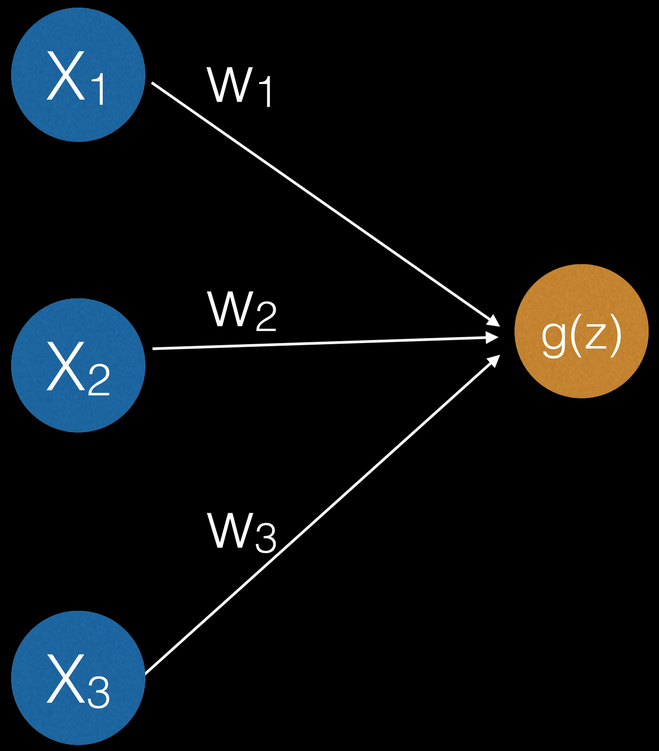


- X_n are data
- w_n are the weights
- z is weighted input. z = WX + b, b is the bias
- g(z) is the activation (more on those later)
- weights and biases can be found with backpropagation
- if g(z) = z this is exactly equivalent to linear regression

Input layer Output layer

single layer neural network

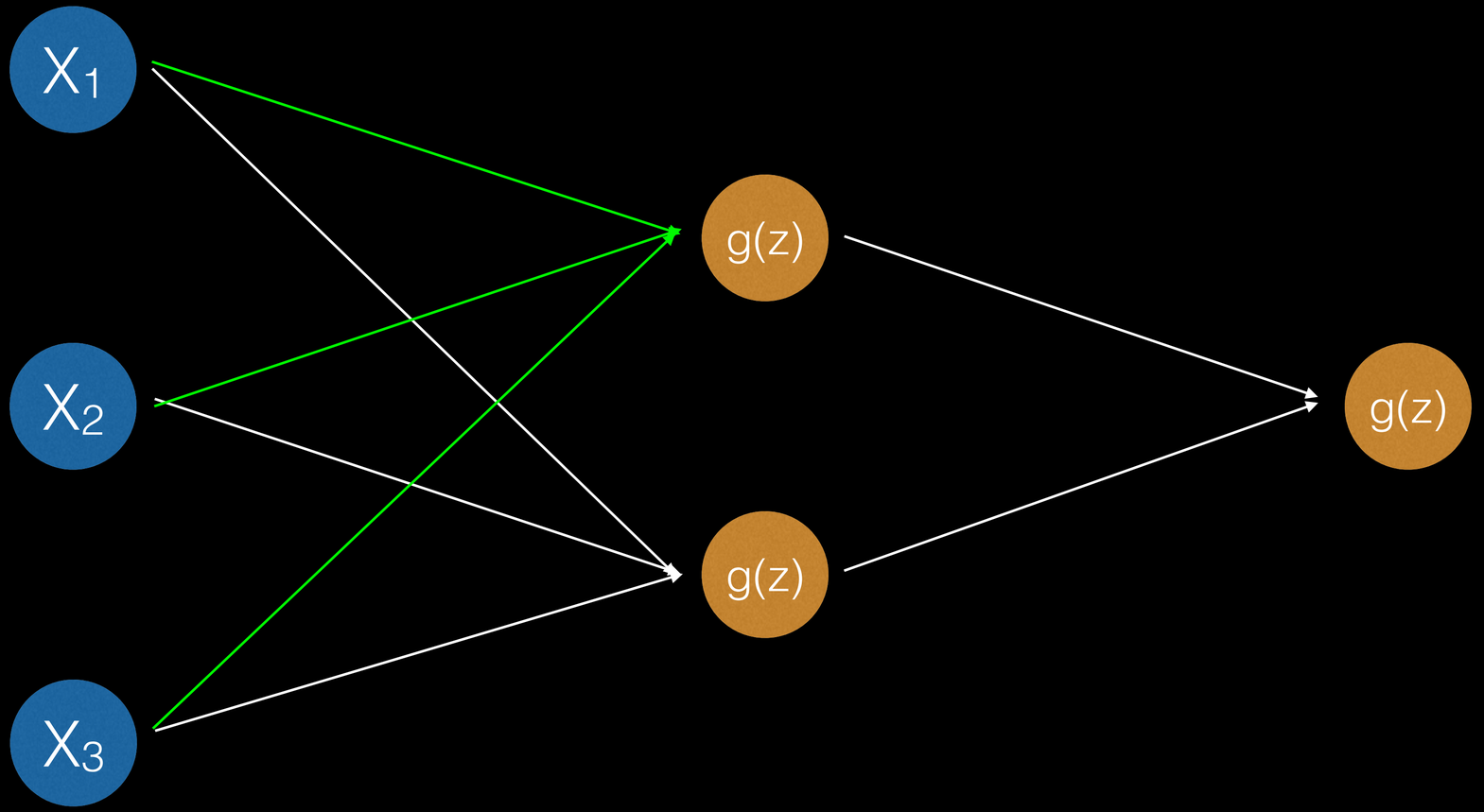
logistic regression



- X_n are data
- w_n are the weights
- z is weighted input. z = WX + b, b is the bias
- g(z) is the activation (more on those later)
- weights and biases can be found with backpropagation
- if g(z) = σ(z) this is exactly equivalent to logistic regression

Input layer Output layer

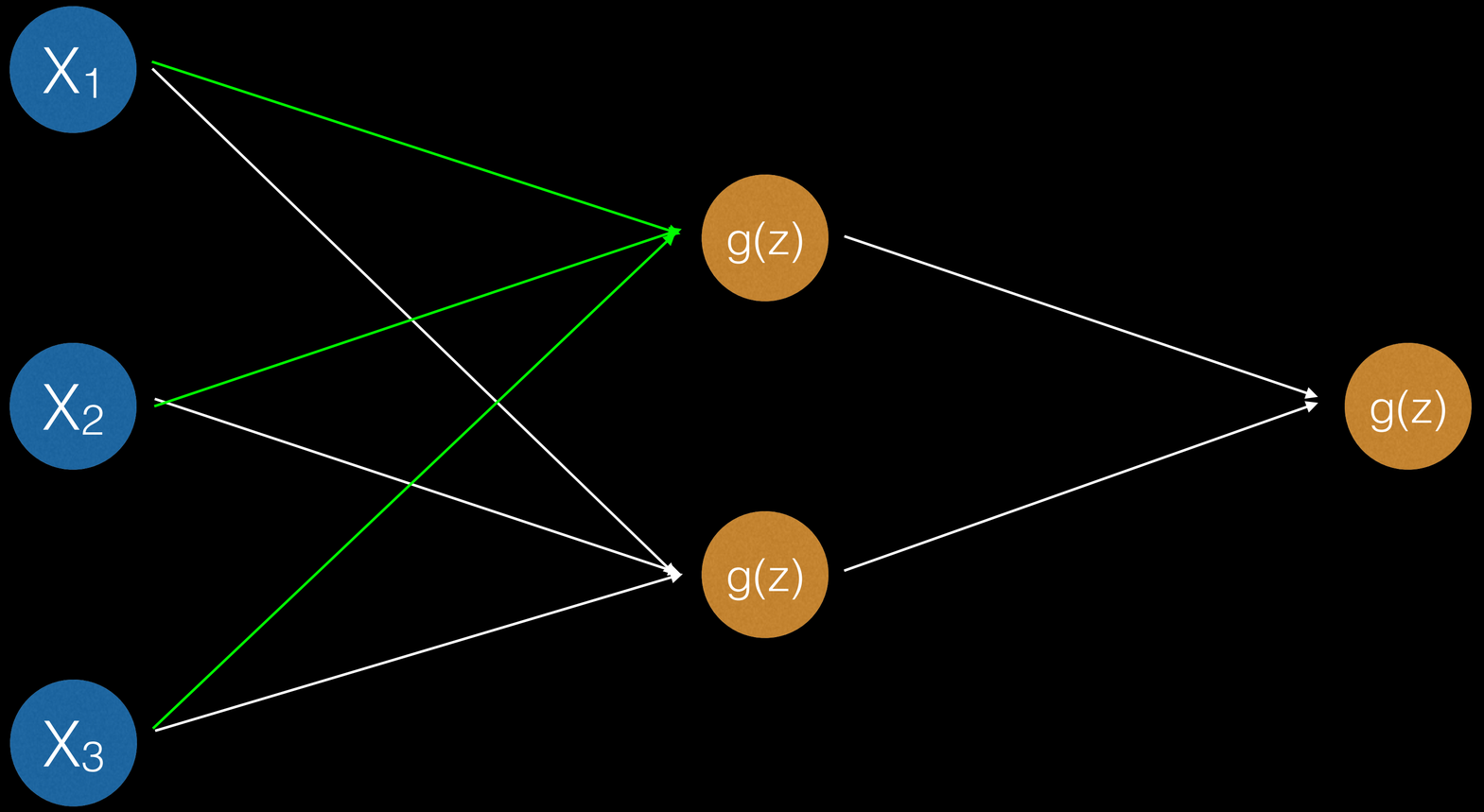
multi layer neural network



input layer

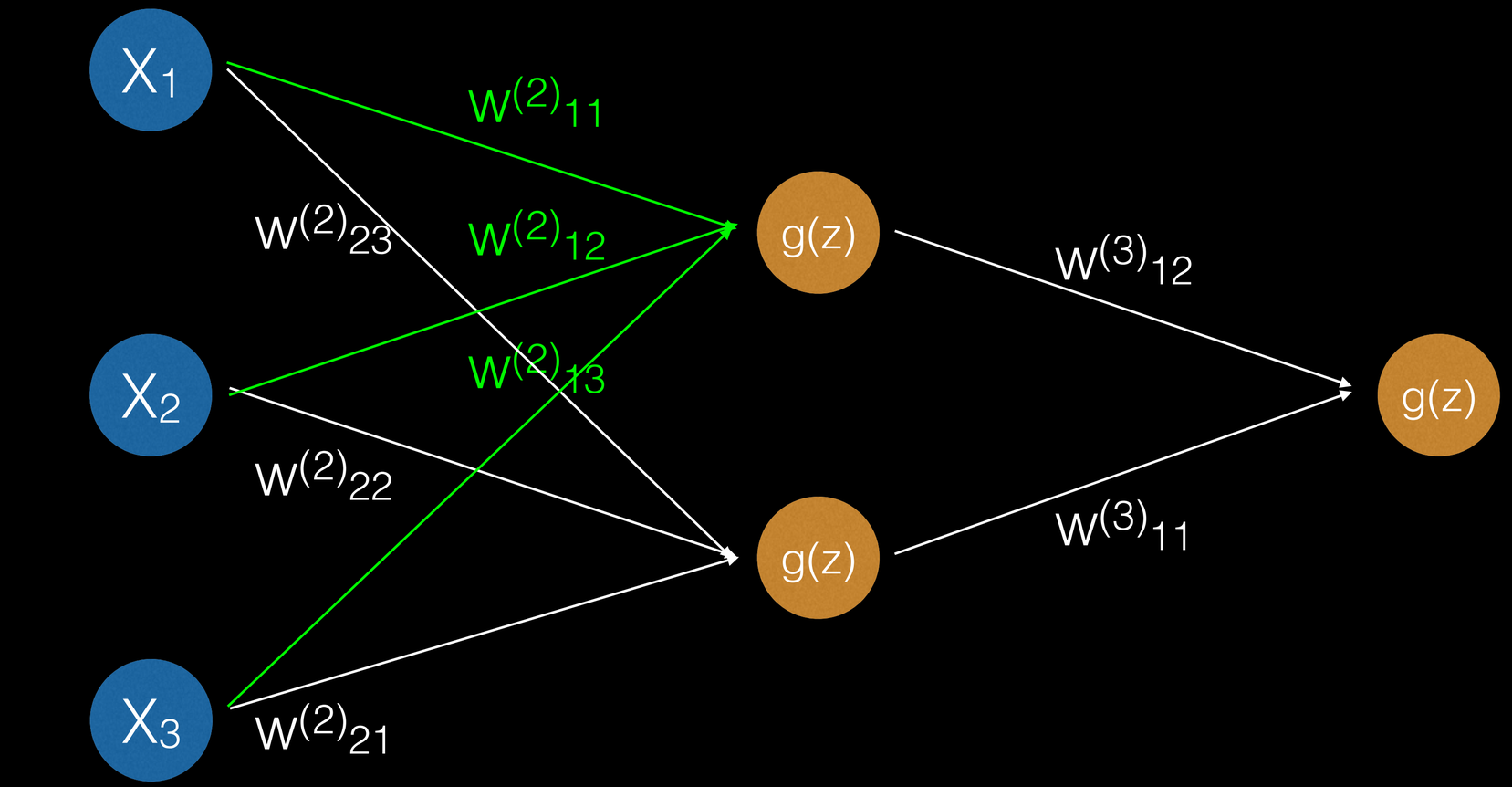
output layer

multi layer neural network



input layer hidden layer output layer

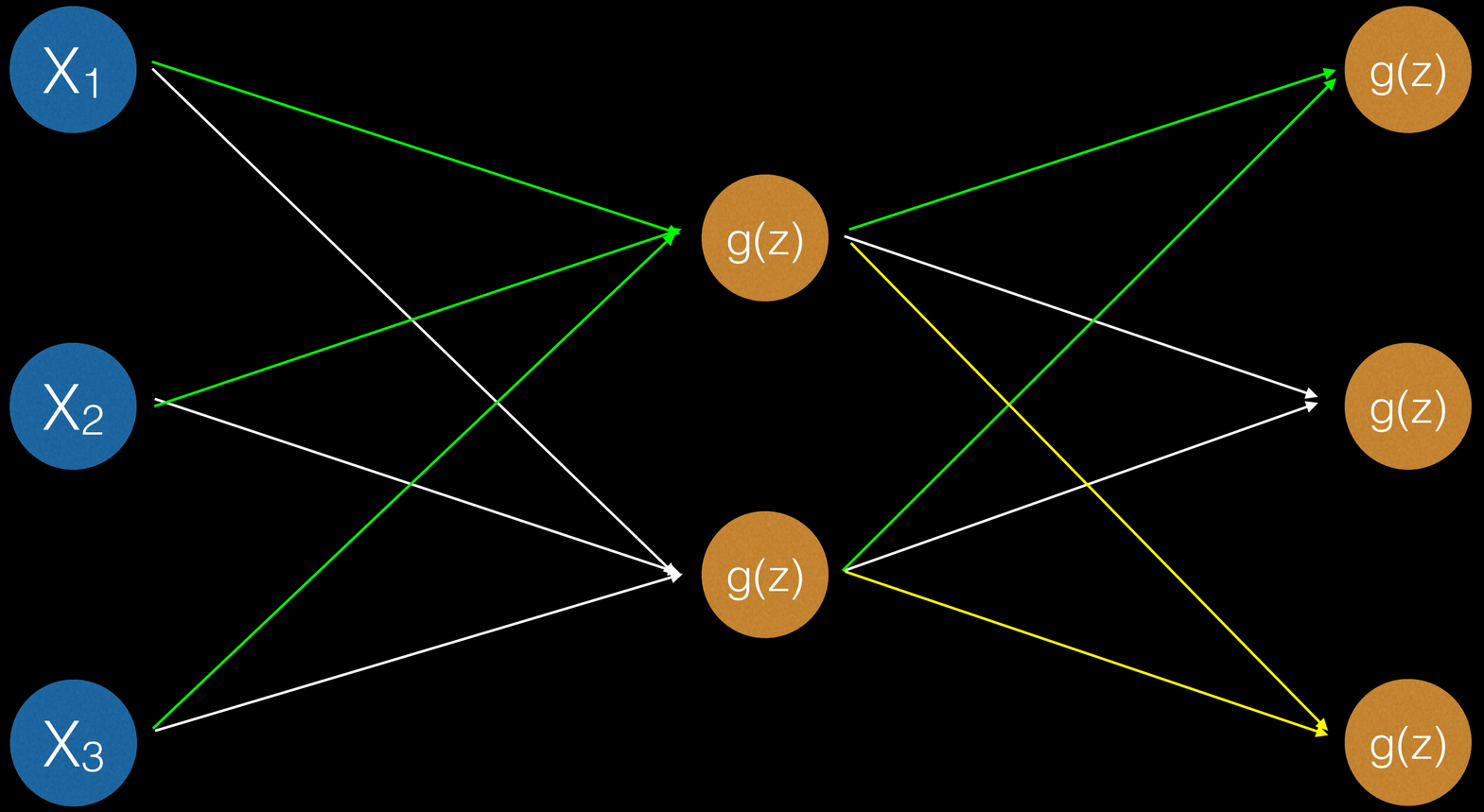
multi layer neural network



input layer hidden layer output layer

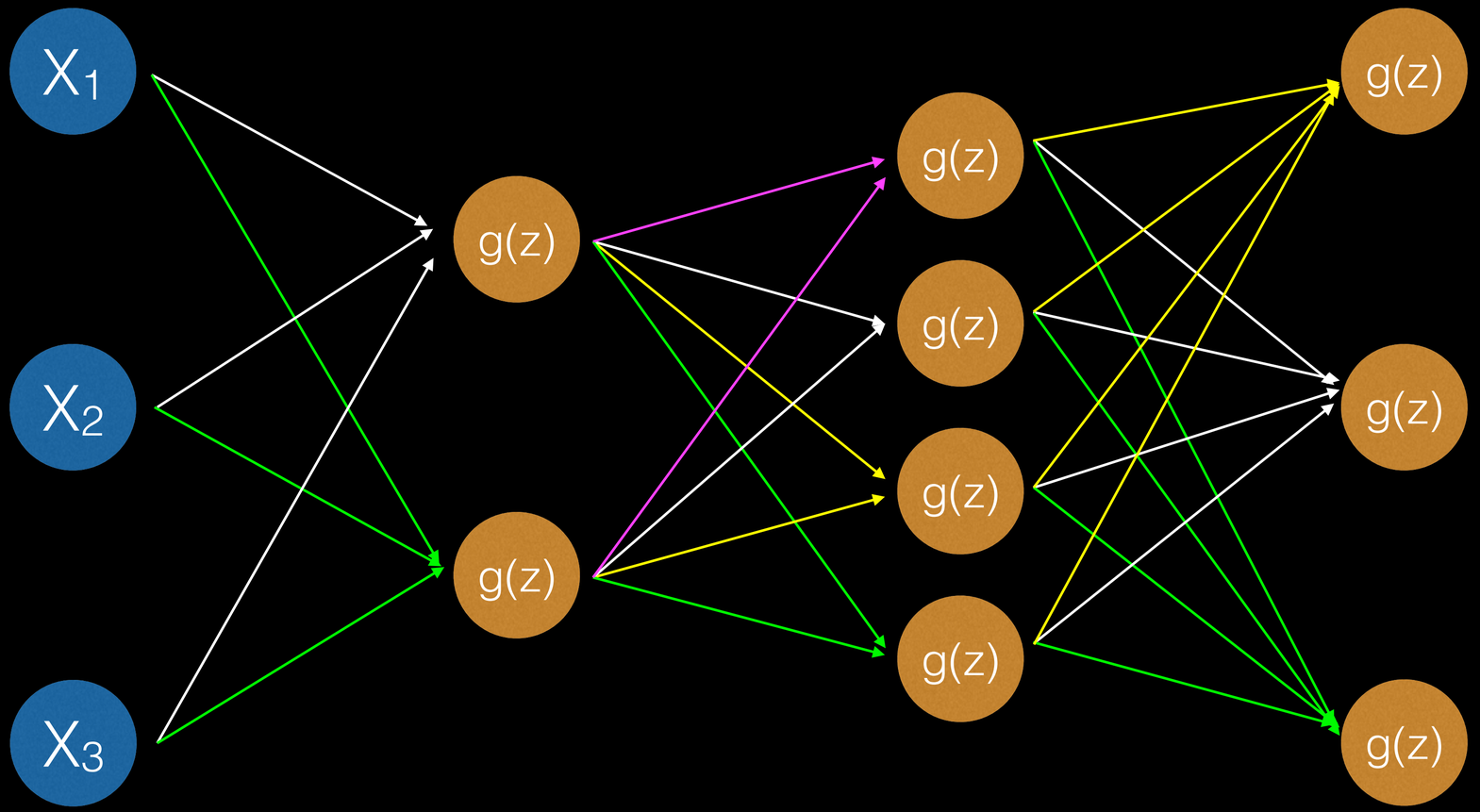
multi layer neural network

you can have multiple outputs
(for classification, and for some other goodness)



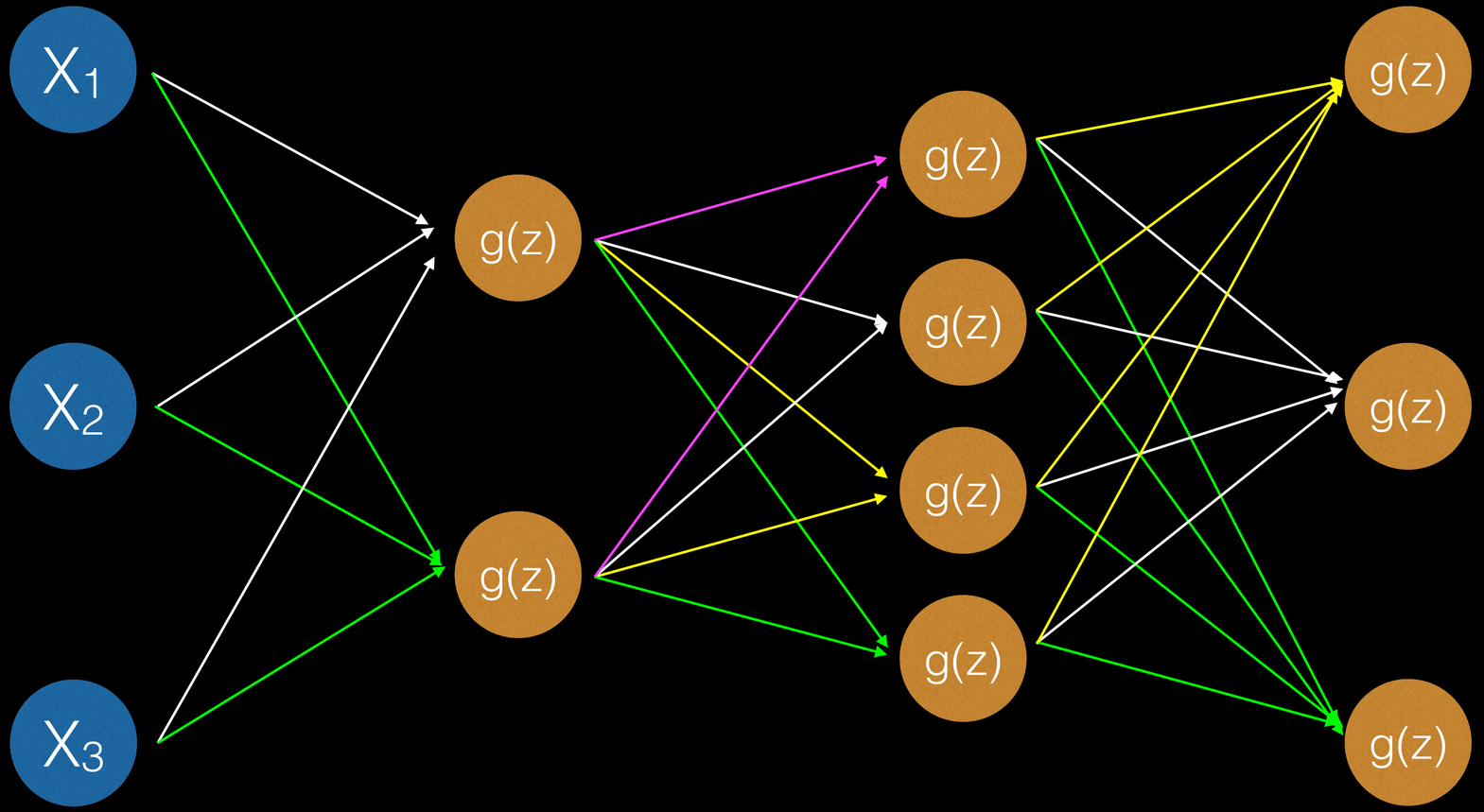
input layer hidden layer output layer

multi layer neural network



multi layer neural network

how many weights in this network?

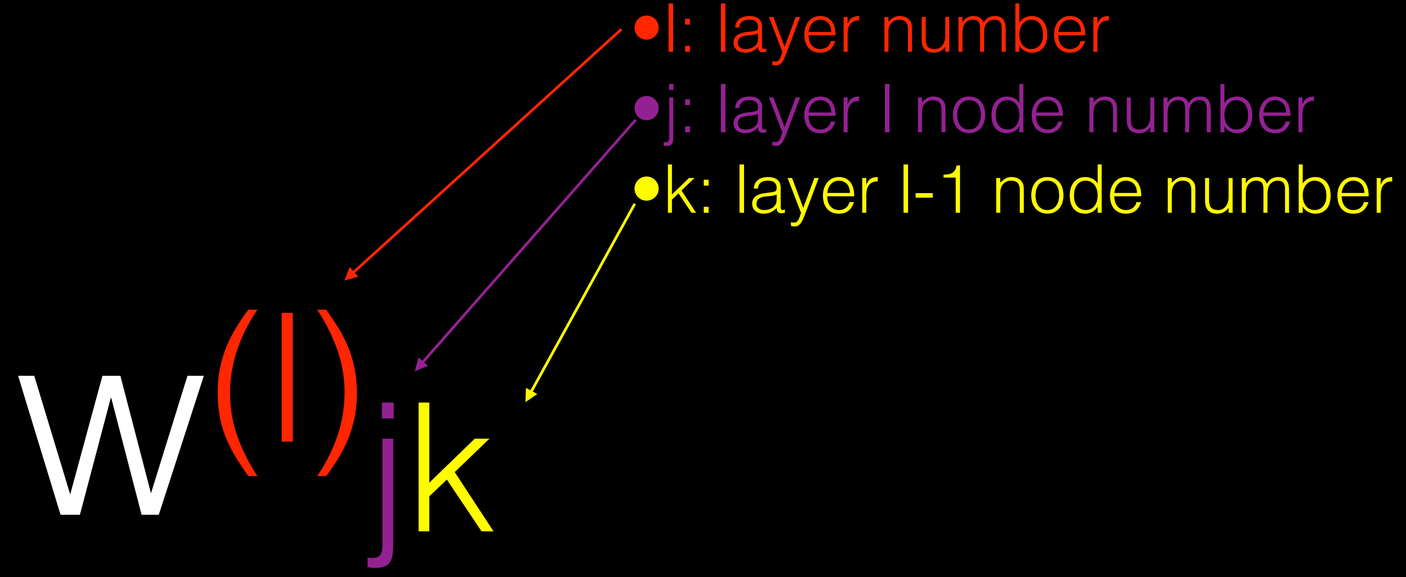


input layer hidden layer1 hidden layer 2 output layer

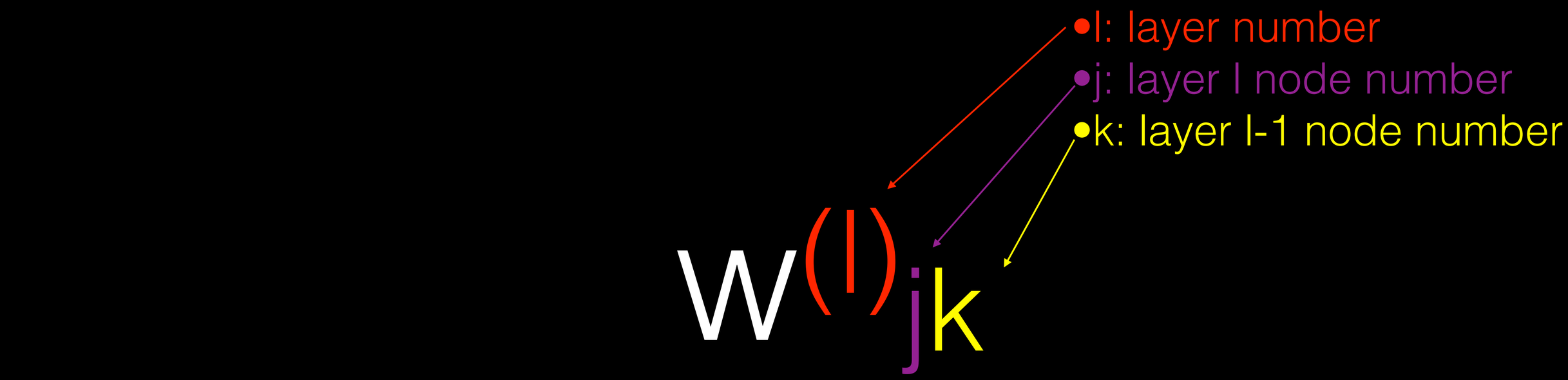
weight notation

$$w^{(l)}_{jk}$$

weight notation



weight notation



weights are mapping from nodes in layer l-1 to nodes in layer l

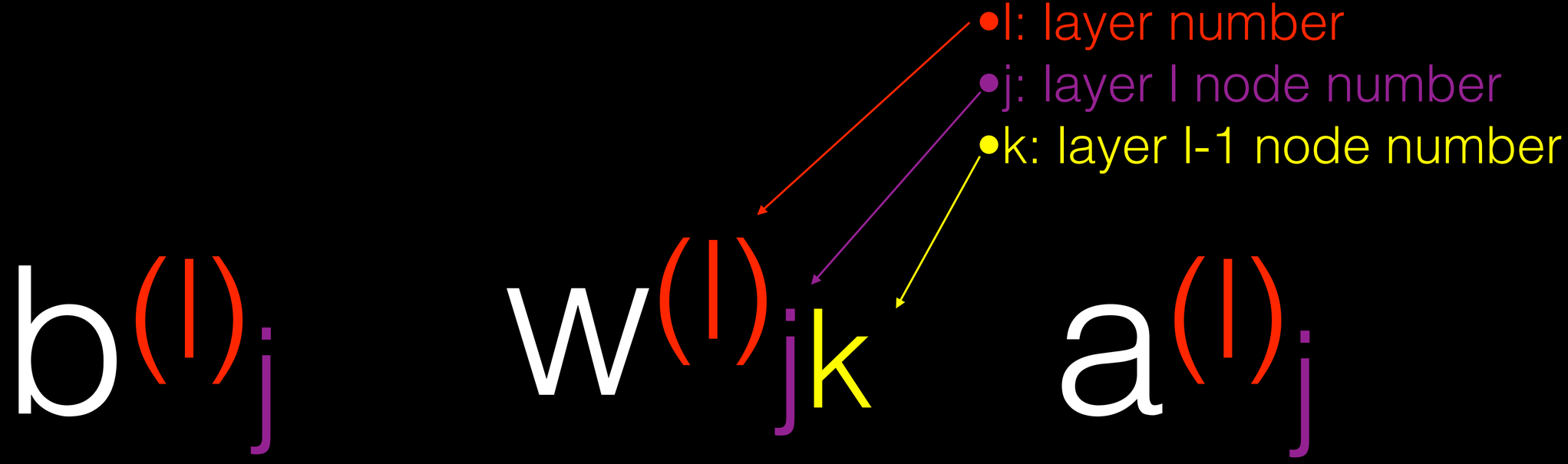
weight notation

- l: layer number
- j: layer l node number
- k: layer l-1 node number

$$b^{(l)}_j \qquad w^{(l)}_{jk} \qquad a^{(l)}_j$$

similar notation for bias and activations

weight notation



similar notation for bias and activations

as always in machine learning, notation is inconsistent
some people will use theta instead of w, others use different notations for the layer and node labels
notation here is taken from http://neuralnetworksanddeeplearning.com/

weight notation

this notation allows the activation of a node to be written as:

$$a_j^l = g(\sum_k w_{jk}^l a_k^{l-1} + b_j^l)$$

or for vector of all nodes in a layer:

$$\vec{a}^{\,l} = g(\vec{w}^{\,l} \bullet \vec{a}^{\,l-1} + \vec{b}^{\,l})$$

$$a_i^1 = X_i$$

weight notation

this notation allows the activation of a node to be written as:

$$a_j^l = g(\sum_k w_{jk}^l a_k^{l-1} + b_j^l)$$

or for vector of all nodes in a layer:

$$\vec{a}^{\,l} = g^l(\vec{w}^l \bullet \vec{a}^{\,l-1} + \vec{b}^{\,l})$$

$$a_i^1 = X_i$$

training

- updates are done with gradient descent
- neural networks are trained using a method called backpropagation
- you first calculate the output of the network with a forward pass, then you run calculations of the error and gradient backwards through the network to update weights
- i am not going to go into detail on the method, but there is a good explanation/derivation of it at: http://neuralnetworksanddeeplearning.com/

training

- in general stochastic gradient descent will be much faster than regular gradient descent
- additional methods for speeding up convergence are used
- a common one is called momentum
- other tricks are often implemented internally in nn frameworks

training

$$v_{t+1} = \mu v_t - \alpha \bigtriangledown L(a_t)$$

$$w_{t+1} = w_t + v_{t+1}$$

training

velocity

$$v_{t+1} = \mu v_t - \alpha \nabla L(a_t)$$

$$w_{t+1} = w_t + v_{t+1}$$

training

velocity

gradient descent step

$$v_{t+1} = \mu v_t - \alpha \nabla L(a_t)$$

$$w_{t+1} = w_t + v_{t+1}$$

training

momentum

velocity

gradient descent step

$$v_{t+1} = \mu v_t - \alpha \nabla L(a_t)$$

$$w_{t+1} = w_t + v_{t+1}$$

activation functions

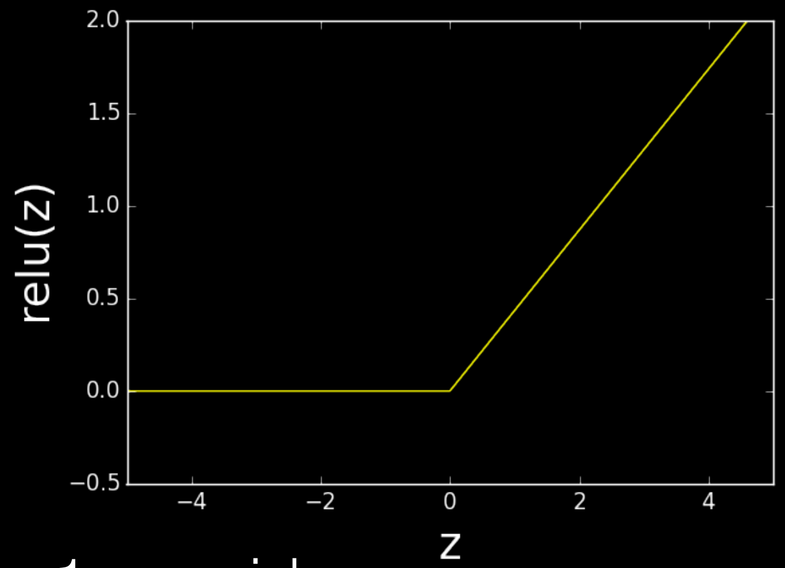
activation functions

- activation functions are the “magic” in neural networks
- transform data space in nonlinear ways, allowing for mapping of complex feature spaces
- in practice just use rectified linear

activation functions

rectified linear (relu (sometimes rel))

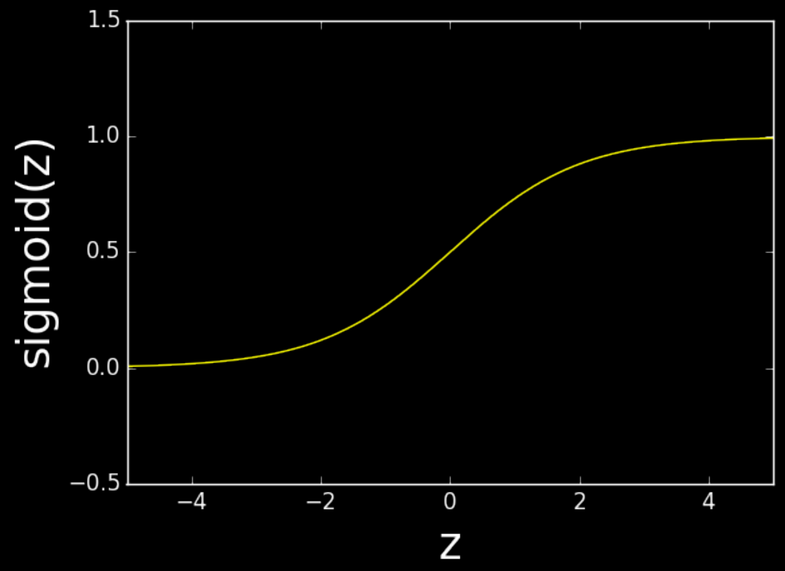
- commonly used activation function
  - it tends to speed up training
  - tends to reduce overfitting
- leaky rectify
  - instead of max(0,x) is max(a*x,x) with a < 1 often <<1. provides some slope for negative values, potentially improving convergence, but still reduces the impact of negative values



activation functions

sigmoid (logistic)

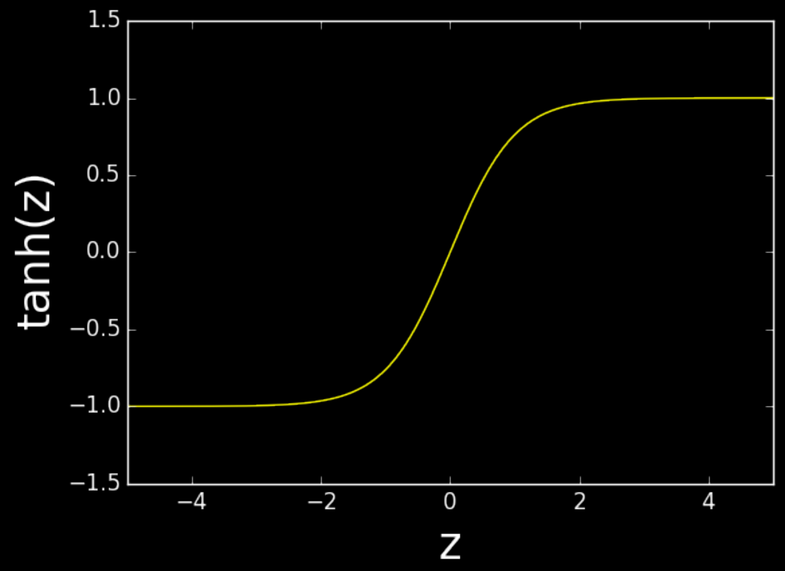
- same function used for logistic regression
  - use it for a two class classification problem
  - can be used to binarize layers
  - not commonly used anymore



activation functions

hyperbolic tangent (tanh)

- like sigmoid but maps to (-1,1) instead of (0,1)
- can used scaled tanh, a*tahn(b*z)
- similar results to sigmoid, but often trains faster



activation functions

other activations

- other common nonlinearities
  - linear
    - just pass the value right through, useful for regression
  - softmax
    - you may remember this from bayesian bandit, useful for multiclass classification
    - provides a probability distribution over the nodes

thats it

- that is the basics of neural networks
- they are just series of dot products with nonlinearities applied
- how they get complicated (and how they are used) is by using more complex layer structures and network architectures

use cases

- current hotness:
  - convolutional networks
    - maintain locality in network connections, very useful for images/ video, certain types of audio, and a few other cases
  - recurrent network
    - layers that have feedback loops, creating directed cycles
    - very useful for a wide range of problems, can be combined with other neural network techniques

use cases

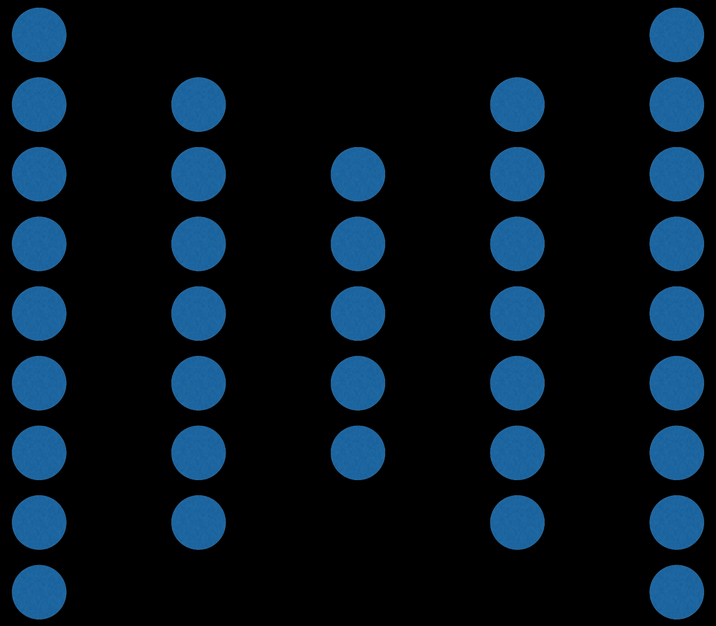
- old goodness:
  - deep belief networks
    - it took me forever to almost figure these out, not sure i can explain them
    - useful for unsupervised feature learning
  - multilayer perceptron
    - more or less equivalent to the example network i drew earlier, it is not commonly used, but is the basis for essentially all modern neural network

use cases

- autoencoders
  - an unsupervised feature extraction network

use cases

- autoencoders
  - an unsupervised feature extraction network



frameworks

- for python, most neural net frameworks are built on theano, which is a package maintained by a machine learning group at the university of toronto
- google recently released tensorflow which functions in a similar way to theano
- both allow you to build computation graphs which are lazy, then evaluate them all at once
- the evaluation is all ported out to C or cuda based libraries which will allow for faster execution

frameworks

- you can write your own neural nets using basic calculations in theano
- there are packages which have prebuilt layer constructs allowing for easier construction of neural network models
- for the sprint you will use lasagne, which has a number of prebuilt nonlinearities, layers, and loss functions
- using nolearn (which extends lasagne with an sklearn type interface) can make training/evaluating networks with lasagne easier
- keras is another package similar to lasagne
- other frameworks
  - deeplearning4j (java based, developed by adam gibson who is probably ~100 feet away from us right now)
  - caffe (framework developed at berkeley, uses gpu's for fast evaluation (this can be done in most of these other packages as well))

how do i do anything with this

- best practices for neural nets are ambiguous and hard to find
- everyone says different things
- if you really want to get into this find someone who knows more about it than you do and pester them
- there is tons of good information out there (check the readme in the repo for starter links)
- neural nets do not currently make machine learning easier