

# Tools & Models for Data Science

## Deep Neural Networks (1)

Chris Jermaine & Risa Myers

Rice University

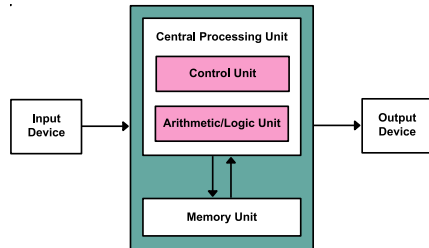


# What Is a Deep NN?

- At highest level
  - It is a nonlinear function
  - Often represented as a graph

# Some History: 1950s

- Interest in artificial NNs started in the 1950's
  - When simple variants were first proposed as a computational model
  - Motivated by the human brain
  - The idea was to model an animal brain
  - Where neurons are either on or off
- Interest in artificial neural networks dwindled as the von Neumann architecture won out

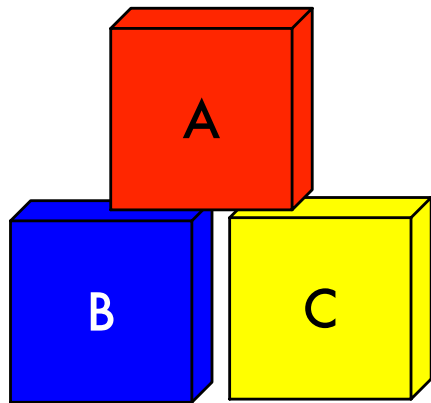


# Some History: 1970s

- Then back-propagation was invented in 1975
  - Effectively, it's using gradient descent to train a NN
  - Given examples, learn a model
- Took ANN further from biological roots
- But made them practically usable

# Some History: 1980s

- 1980's: the computers couldn't handle the computations
- 2nd "AI winter" 1987 - 1993
  - No serious progress
  - Government felt that scientists were lying to them



# Some History: 1990s

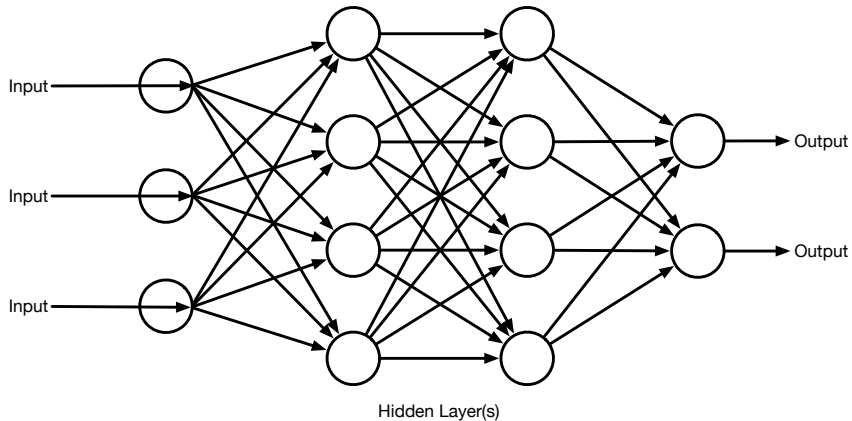
- Interest was renewed, mainly via statistical methods
  - As methods such as SVMs gained popularity
  - Kernel trick surfaced around 1992
- By late 1990's ANNs were almost seen as a joke
  - No one who was hip and with it actually looked at them
  - Why?
  - Way too data hungry
  - Way too computationally expensive to train
  - Alternatives (e.g. SVM) worked pretty well

# Some History: 2000s

- But by the end of the aughts, that changed
  - We had tons of data (Google!)
  - We have tons of computational ability
  - Even though Moore's law has stalled out, we have multi-core, GPUs and the like
  - And ANNs are very amenable to parallelization
- Suddenly, ANNs were doing things that amazed everyone
  - Massive increase in accuracy in speech, image recognition
  - Learning to play Go, beat the best players in the world
  - AlphaGo Zero: Learned by playing games against itself



# What Does a NN Look Like?

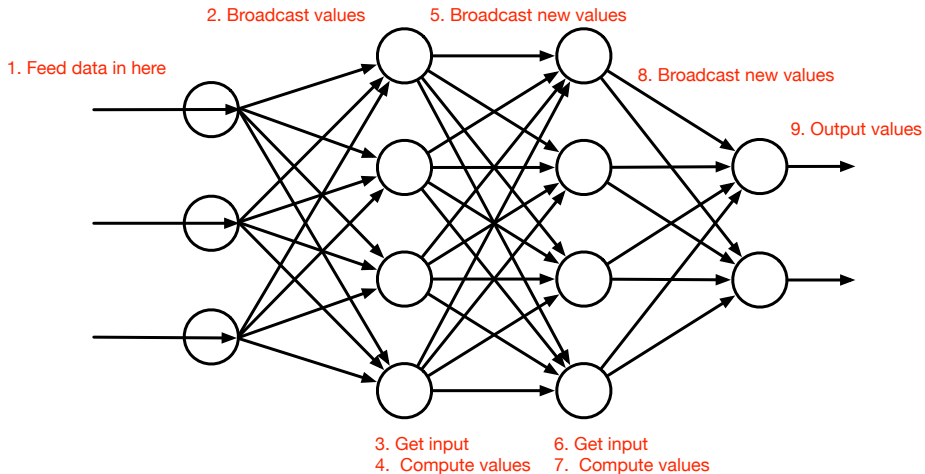




# What Does a NN Look Like?

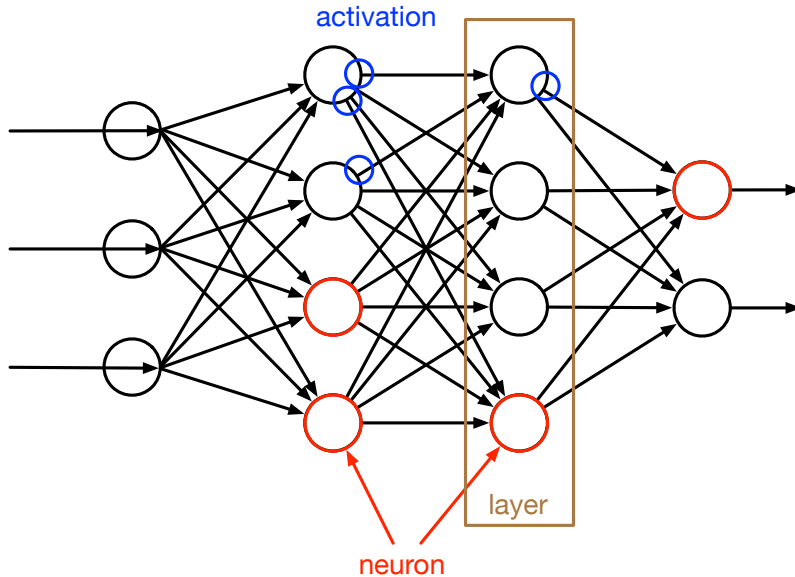
- Inputs
  - Often floating point values
  - Could be a multidimensional feature vector
- Hidden layer(s)
  - Receive input
  - Perform transformation
  - Pass values on
  - Often fully connected
- Outputs
  - Often probabilities
  - Reflect levels of certainty

# What are the Steps in NN Processing?



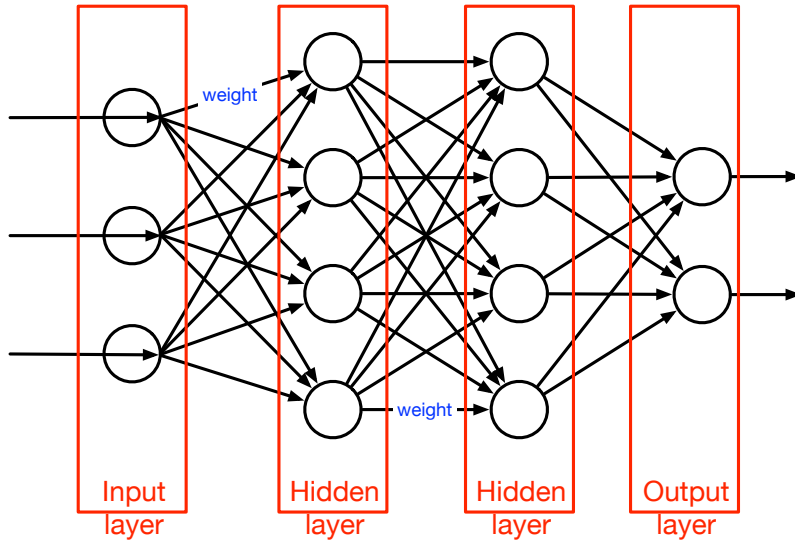
- The circles are called “neurons”
  - Are little computational units
  - Function they compute is always chosen to be differentiable
  - Since learning is via gradient descent
- Output of a neuron is called an “activation”
- Neurons are organized into “layers”
  - In simplest case, layers take input only from preceding layer

# NN Terminology



- Top/right layer in deep NN is the “output layer” (layer  $t$  for “top”)
- Bottom/left layer in deep NN is the “input layer” (layer 1)
  - Activation of input neurons read directly from input features
- Are one or more “hidden layers” between those two (layers  $1 < i < t$ )
- Each link between neurons has a “weight”
  - Input into a neuron is the weighted sum of activation that travels over all edges that point into the neuron

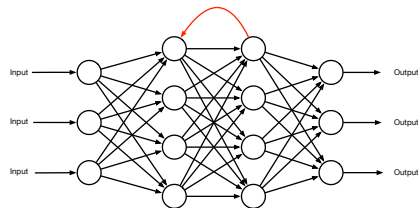
# NN Terminology



- Today we'll assume a “fully connected” NN
  - That is, each neuron in layer  $i > 1$  takes input from all neurons in layer  $i - 1$
- But there are other connection patterns between layers
  - Example: convolutional layers
  - Used extensively in image processing
  - Where subsets of neurons are constrained to only work on contiguous regions of input data

# Simplest NNs

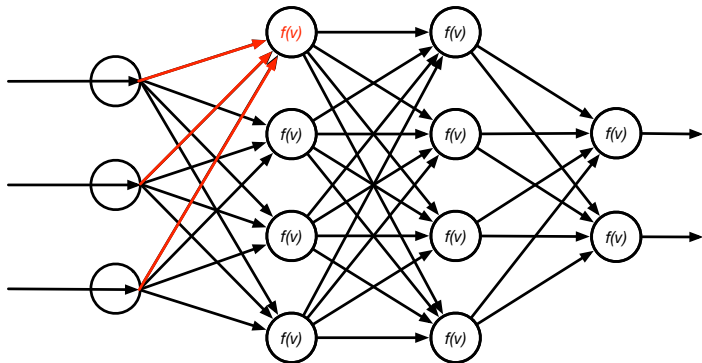
- Are called “feed-forward” NNs
  - Because activations only flow forward thru network
  - Flow from input layer
  - Thru hidden layers
  - And out of the output layer
- Other options include “recurrent” NNs
  - In an RNN, input from upper layer fed recursively into input in lower layer





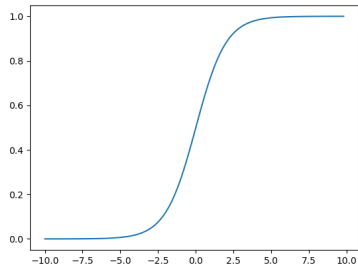
# What Function Does Each Neuron Compute?

- Need each neuron to compute some function  $f(v)$ 
  - $v$  is weighted sum over input activations
  - $f(v)$  is often simple and differentiable



# What Function Does Each Neuron Compute?

- Typically a “sigmoid” function
  - So neuron goes from “off” to “on” (like a soft binary gate)
  - Output is a floating point number
  - But smoothly, so differentiable (gradient descent once again!!)



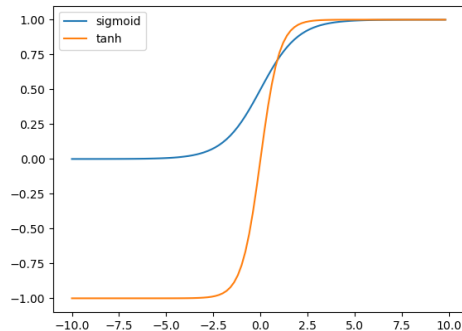
# Classic Sigmoid Functions

- First, the hyperbolic tangent function

$$y = \tanh(v)$$

- Second, the logistic function

$$y = (1 + e^{-v})^{-1}$$

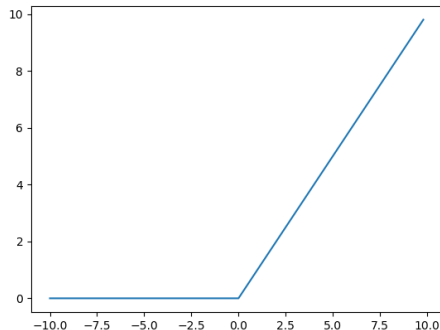


# Modern Sigmoid Function

- ReLU “Rectified linear unit”

$$y = \max(0, v)$$

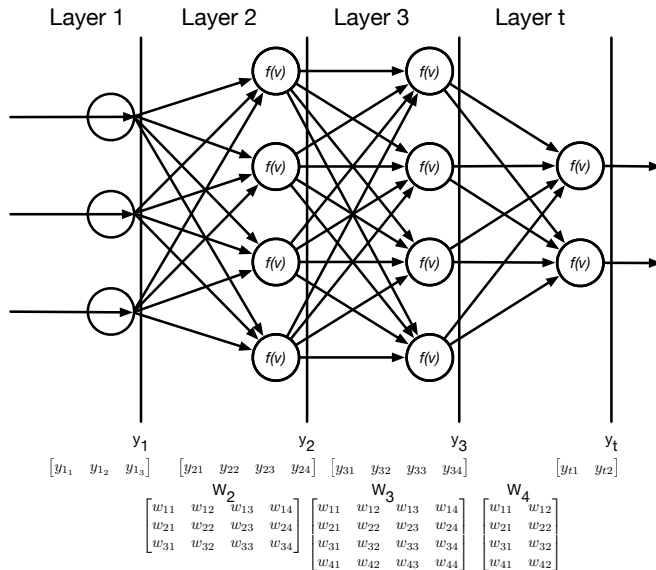
- 0 or pass through the input value



# Evaluation of a Feed-Forward NN

- Typically done via vectorized operations... fast!
- How to push activation thru a network...
  - Let  $y_i$  be a row vector denoting output of  $i$ th layer
  - Define  $y_1 = x$ , the input data
  - Let  $f$  denote the vector-valued function applying the activation function to each item in a vector
  - Let  $W_i$  denote the matrix of weights connecting layer  $i - 1$  to layer  $i$
  - If layer  $i - 1$  has  $n$  neurons, layer  $i$  has  $m$  neurons, then  $W_i$  has  $n$  rows,  $m$  columns
  - And  $y_i = f(y_{i-1} \times W_i)$

# NN Evaluation



# Evaluation of a Feed-Forward NN

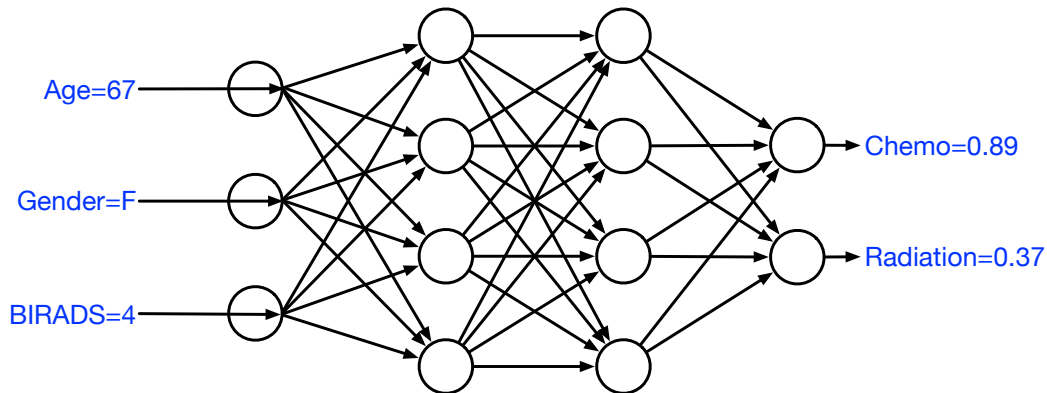
- It's all about matrix & vector multiplication
- You could write lots of loops in Python
- Recall, we've talked a lot about vector operations
- GPUs are really good at linear algebra
- So, write linear algebra

## In practice

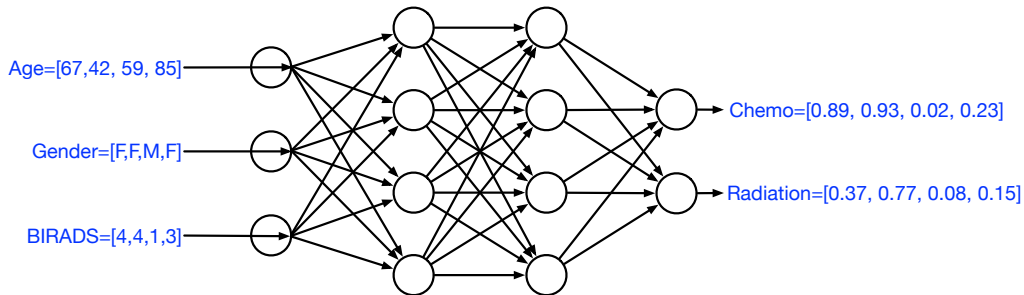
- We send multiple inputs through the network at the same time
- So,  $y$  is a matrix, not a vector
- Why?
  - It's more efficient
  - Theoretically, it's the same complexity, but in practice, it's different



# NN Example



# NN Example with Multiple Inputs



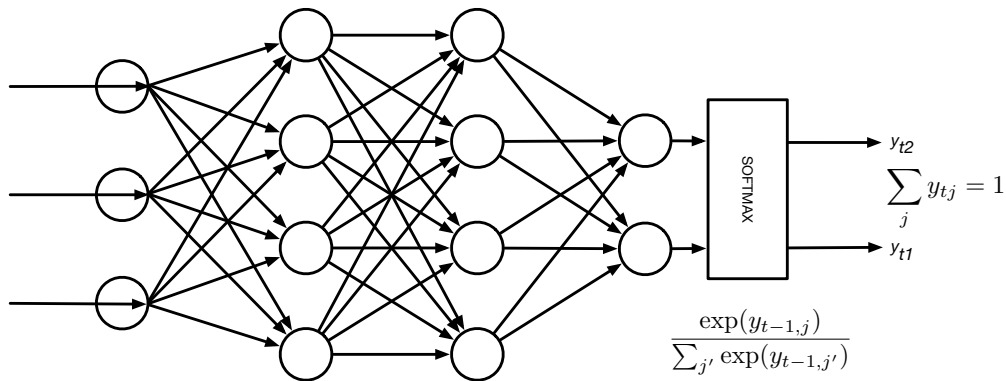
# Producing Outputs

- NNs classically used for classification
  - Output is function of activations of second-to-last layer
- Often, we want certainty - e.g. a probability
- Typically we use a “softmax” function
  - That is, neuron  $j$  at layer  $t$  (the “top” layer) outputs:

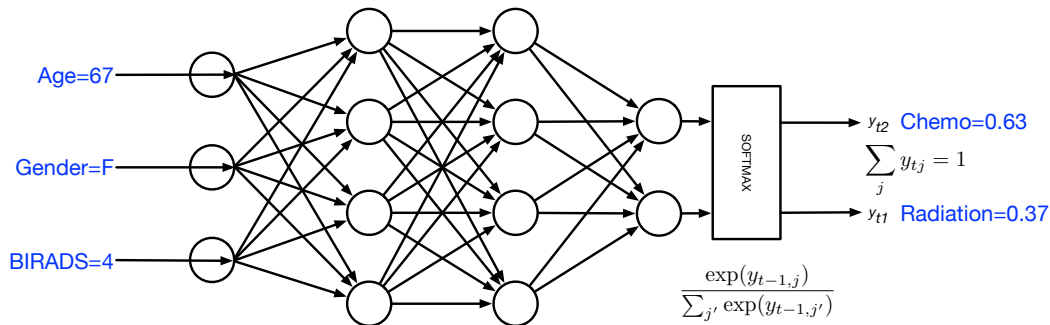
$$\frac{\exp(y_{t-1,j})}{\sum_{j'} \exp(y_{t-1,j'})} = \frac{\text{output of previous layer}}{\text{normalization term}}$$

- Since all  $y_t$  values then sum to one...
  - $y_{t,j}$  typically viewed as the probability that correct class label for data  $y_1$  is  $j$

# NN with Softmax



# NN Example



# Another Example Classification

- Want to solve a classification problem
- Chihuahua or muffin?
- Push the picture through the FF network
- End with two neurons
  - Chihuahua
  - Muffin



# Using Softmax

- Could take the max of the two outputs
- But it's a smooth function – we have degrees of “On”
- The **max** function is not differentiable
- Use **softmax** instead
- Gives us  $\text{Pr}[\text{Chihuahua}]$  and  $\text{Pr}[\text{Muffin}]$
- Sample using probabilities to choose one



# Another Example

- E-Discovery: Want to identify emails that are relevant to a court case
- Say there are 100 topics of interest to the case
- Crowd source relevant example emails for each topic
- Each email might contain any number of the topics
- Goal: identify additional emails that are relevant to each topic





# Yet Another Example

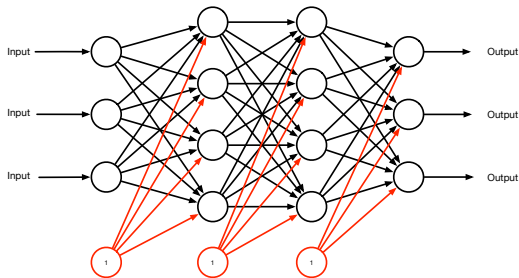
- Want to identify objects in a image
- Each image may contain multiple objects
- Push the pictures through the FF network
- End with a neuron for every object you want to identify
  - Cat
  - Dog
  - Puppy
  - Tree
  - Mug
  - ...
- The output of each neuron gives the confidence of that object being in the image



- Just like in regression....
  - We want to be able to bias output of each neuron
  - So, the neuron is not on/off 50% of the time with totally random input
  - Maybe we want it on just 5% of the time with random input
  - Basically like adding an intercept term
- How to do this?

## ■ Easy!

- Just like in LR
- Have a special neuron at each layer
- It always outputs the value 1
- Regardless of its inputs
- Then the last row of each  $W_i$  will have biases
- Since  $w_{i,j} \times 1$  will be added into input to each neuron in layer  $i + 1$



- What is a NN?
- What are the components of a NN?
- What kinds of problems can it solve?

- What are the weights?
- Accomplished via gradient descent
- Gives rise to the famous “back-propagation” algorithm

# Questions?