Tools & Models for Data Science

Neural Networks (1)

Chris Jermaine & Risa Myers

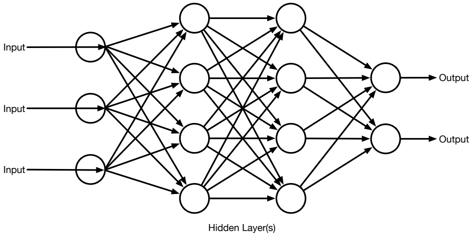
Rice University

Objectives

- Learn what an Artificial Neural Network is
- Learn the history of neural networks
- Learn components of and terminology for neural networks
- Learn the types of problems that can be addressed with neural networks

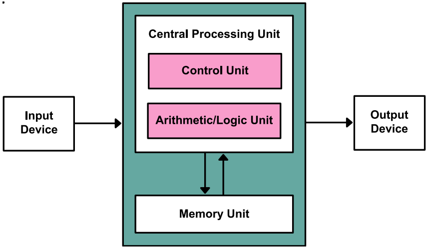
What Is a Neural Network?

- 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



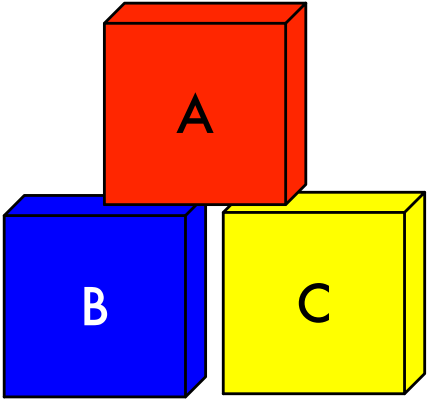
von Neumann architecture

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



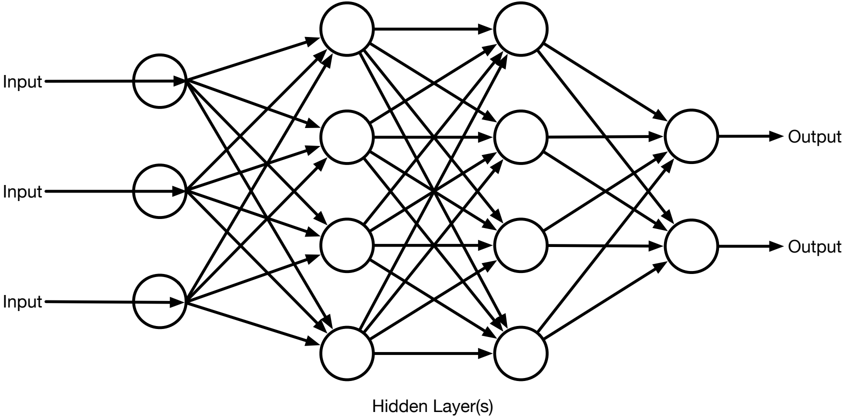
Some History: 1990s

- 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

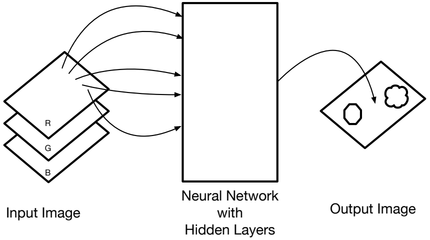
- But by the end of the aughts, that changed
  - We had tons of data (Google!)
  - We have tons of computational ability
  - Even though Moore's law has stalled out, we have multi-core, GPUs and the like
  - And ANNs are very amenable to parallelization
- Suddenly, ANNs were doing things that amazed 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?

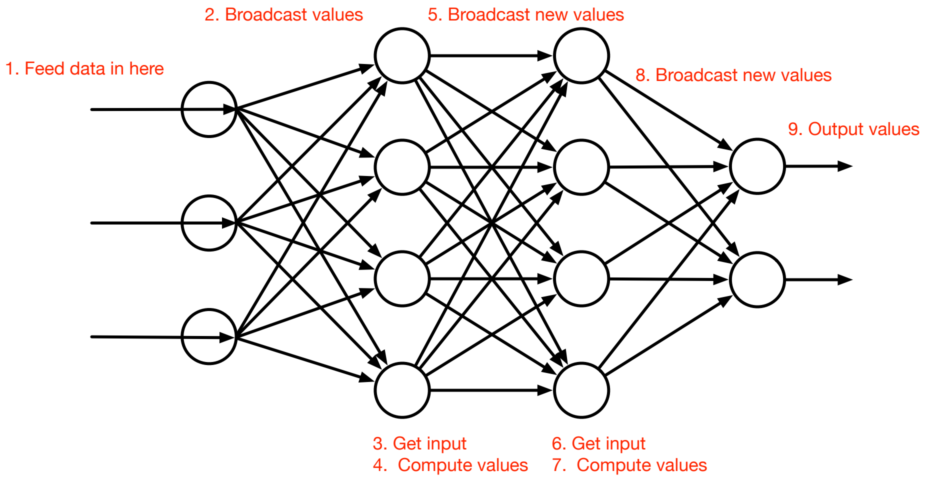


Terminology (1)

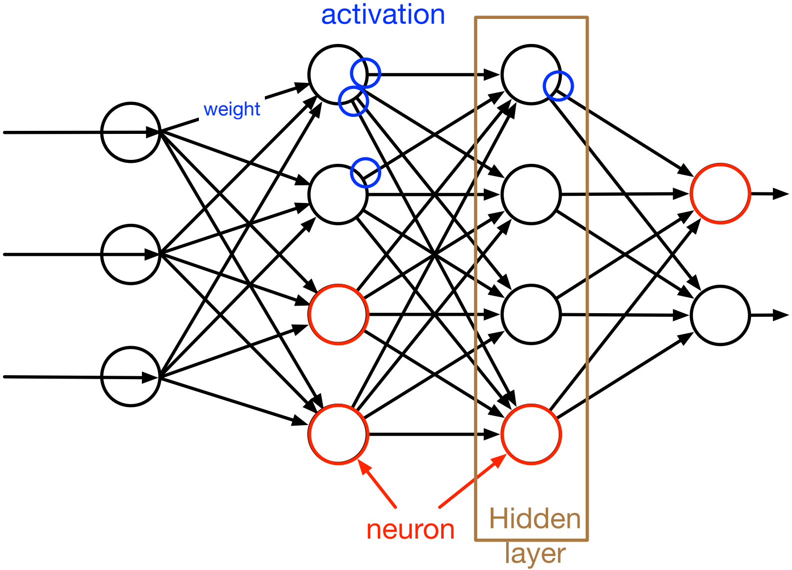
- Inputs
  - Data!
  - # is based on the shape of your input
  - Often floating point values
  - Could be a multidimensional feature vector
  - Could be images
- Hidden layer(s)
  - Receive input
  - Perform transformation
  - Pass values on
  - Often fully connected
- Outputs
  - Often probabilities
  - Reflect levels of certainty
  - # depends on what you want e.g. a probability for each class in a set



What are the Steps in NN Processing?



Terminology (2)

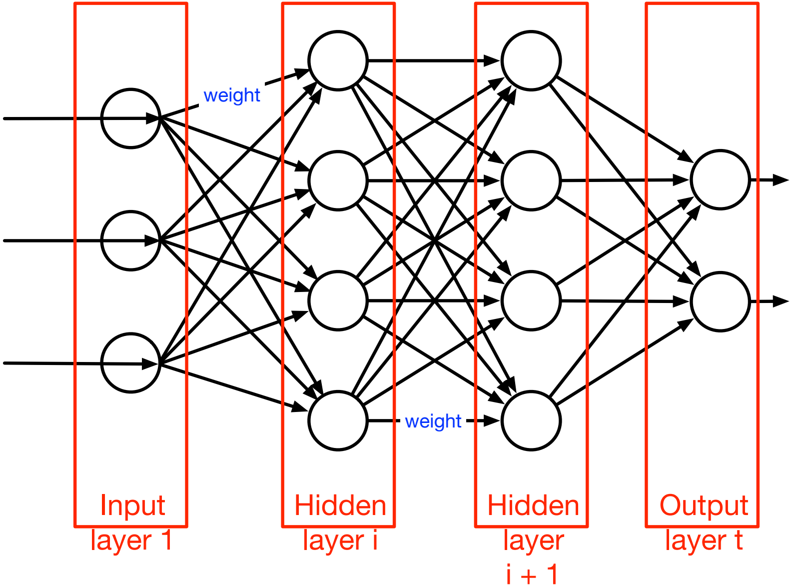


Terminology (2)

- The circles are called “neurons”
  - Little computational units
  - Computed function is usually continuous and “nice”
  - Since learning is via gradient descent [1]
- Output of a neuron is called an “activation”
- Neurons are organized into “layers”
  - In simplest case, layers take input only from preceding layer
  - “Hidden” because they are not visible externally

[1] There are more complex variants

Terminology (4)



Terminology (4)

- **Deep** Neural Networks have $>$ 1 hidden layer
- Top/right layer in a NN is the “output layer” (layer t for “top”)
- Bottom/left layer in a NN is the “input layer” (layer 1)
  - Activation of input neurons read directly from input features
- One or more “hidden layers” are between those two (layers $1 < i < t$)
- Each link between neurons has a “weight”
  - Input into a neuron is the weighted sum of activations that travel over all edges that point into the neuron

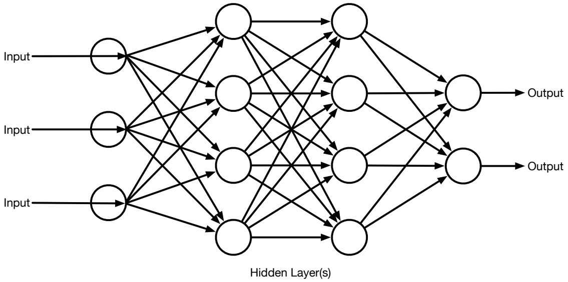
Neural Network Types

- 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 [2]
  - Convolutional layers
    - Used extensively in image processing
    - Where subsets of neurons are constrained to only work on contiguous regions of input data
  - Recurrent Neural Networks (RNNs)
    - In an RNN, input from upper layer fed recursively into input in lower layer
    - Input to a layer can come from $>$ 1 layer

[2] later lectures

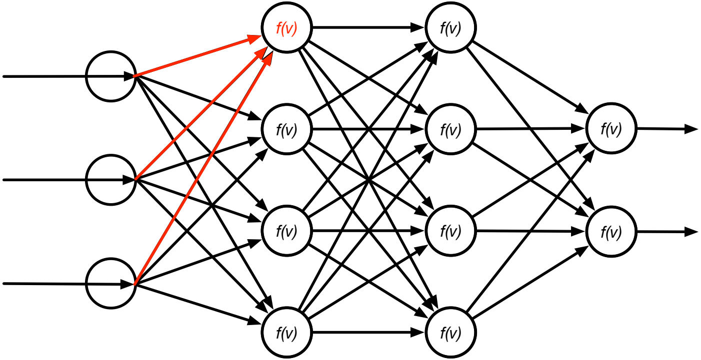
Simplest NNs

- 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

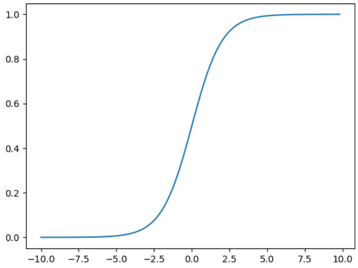


What Function Does Each Neuron Compute?

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



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



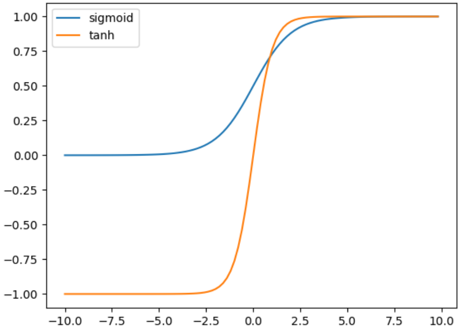
Classic Sigmoid Functions

- First, the hyperbolic tangent function

$$y = \tanh(v)$$

- Second, the logistic function

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

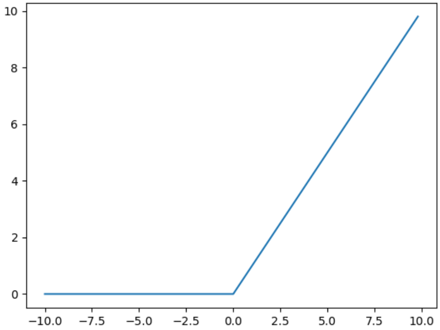


Modern Activation Function

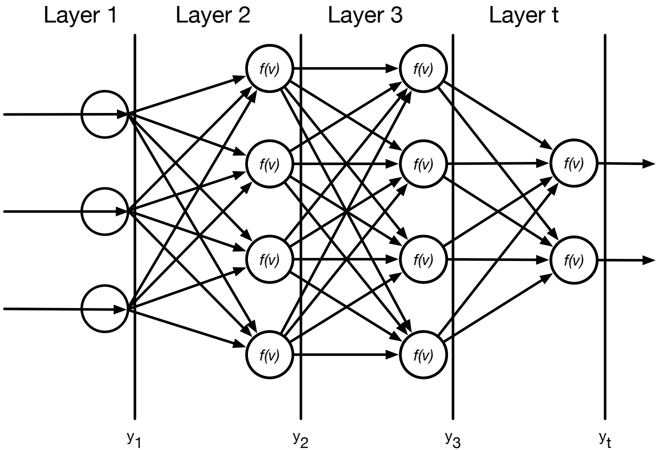
- ReLU “Rectified linear unit”

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

- 0 or pass through the input value



NN Evaluation



$[y_{1_1} \quad y_{1_2} \quad y_{1_3}]$ $[y_{21} \quad y_{22} \quad y_{23} \quad y_{24}]$ $[y_{31} \quad y_{32} \quad y_{33} \quad y_{34}]$ $[y_{t1} \quad y_{t2}]$

$$W_2 = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \end{bmatrix} \quad W_3 = \begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \\ w_{41} & w_{42} & w_{43} & w_{44} \end{bmatrix} \quad W_4 = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \\ w_{41} & w_{42} \end{bmatrix}$$

Evaluation of a Feed-Forward NN

- Typically done via vectorized operations... fast!
- How to push activation thru a network...
  - Let y_i be a vector denoting output of ith 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 $y_i = f(W_i \otimes y_{i-1})$, the tensor product of W_i and y_{i-1}

$$\begin{bmatrix} & & \\ & W & \\ & & \end{bmatrix} \otimes \begin{bmatrix} \\ y_{i-1} \\ \\ \end{bmatrix} = \begin{bmatrix} \\ y_i \\ \\ \end{bmatrix}$$

Evaluation of a Feed-Forward NN

- It's all about matrix & vector multiplication
- ... and applying a non-linear activation function
- 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

The input

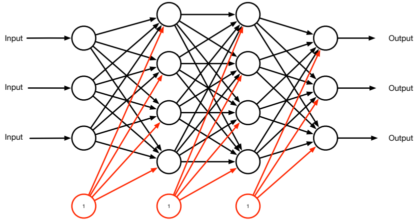
In practice

- 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

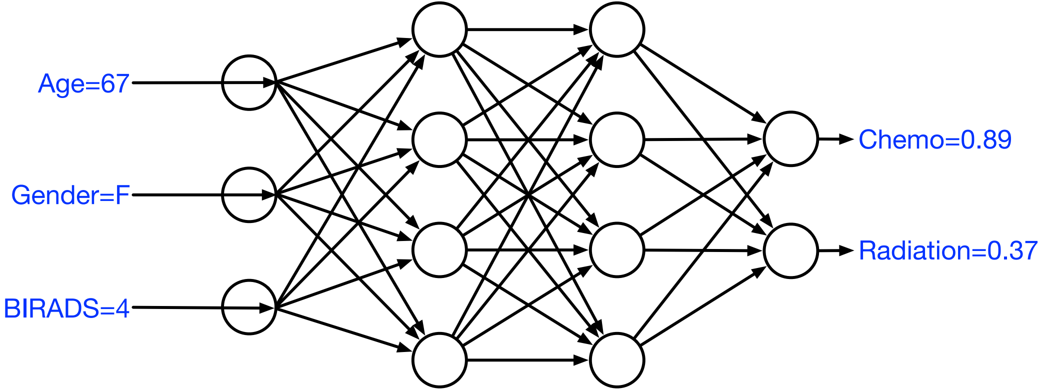
Bias

- Just like in Linear Regression
- Have a special neuron at each layer
- It always outputs the value 1, regardless of its inputs
- Recall i is the layer and j is the neuron index
- 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 $i+1$
- $y_i = f(W_i \otimes y_{i-1} + b_i)$
- b_i is a scalar that is added to every entry:

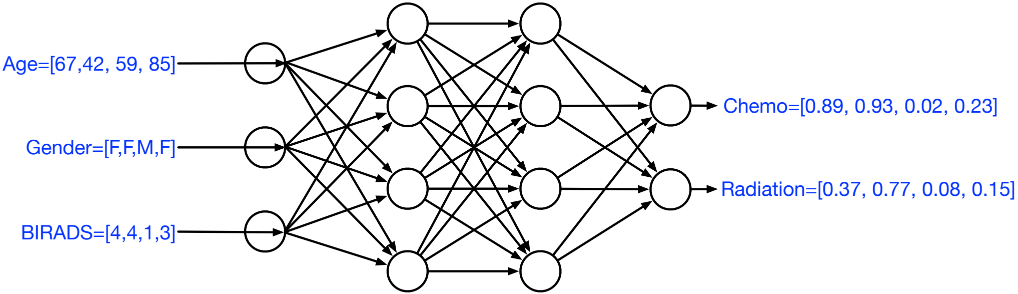
$$\begin{bmatrix} & & \\ & W & \\ & & \end{bmatrix} \otimes \begin{bmatrix} \\ y_{i-1} \\ \\ \end{bmatrix} + b_i \begin{bmatrix} 1 \\ . \\ . \\ . \\ 1 \end{bmatrix} = \begin{bmatrix} \\ y_i \\ \\ \end{bmatrix}$$



NN Example



NN Example with Multiple Inputs



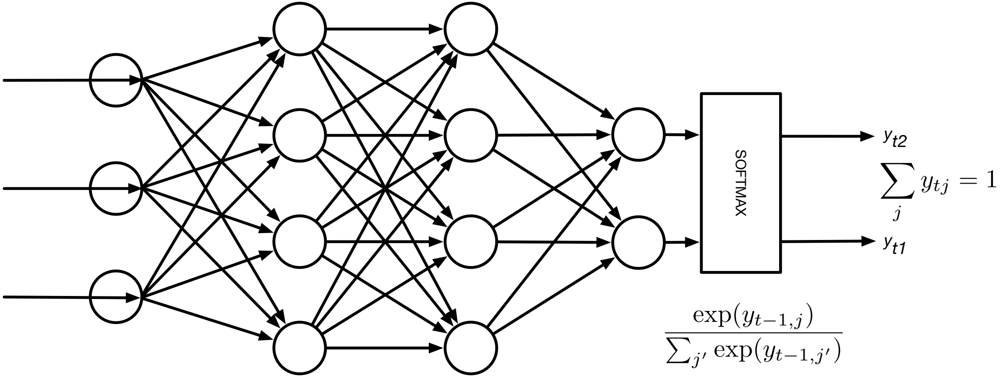
Producing Outputs

- NNs classically used for classification
  - Output is function of activations of the last hidden 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:

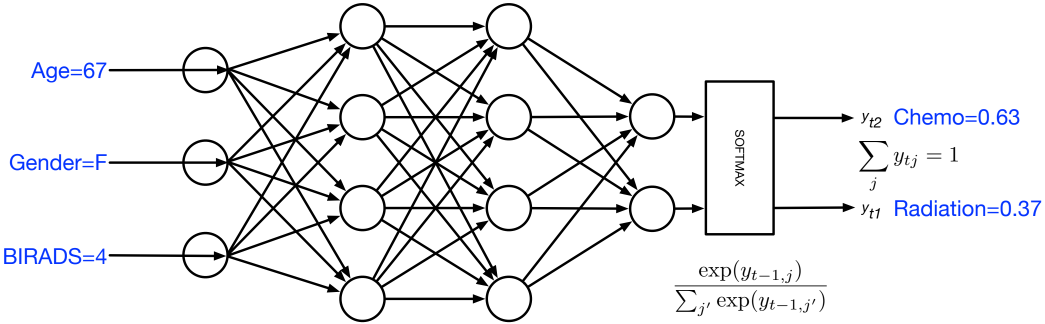
$$\frac{\exp(y_{t-1,j})}{\sum_{j'} \exp(y_{t-1,j'})} = \frac{\texttt{output of previous layer}}{\texttt{normalization term}}$$

- 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


Age=67
Gender=F
BIRADS=4
SOFTMAX
y_{t2} Chemo=0.63
$\sum_j y_{tj} = 1$
y_{t1} Radiation=0.37
$\frac{\exp(y_{t-1,j})}{\sum_{j'} \exp(y_{t-1,j'})}$

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

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 fo each topic
- Each email might contain any number of the topics
- Goal: identify additional emails that are relevant to each topic

- Amazon
- Search
- Profitability
- Relevance
- Third-party sellers
- Product recommendations

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

Summary

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

Learning

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