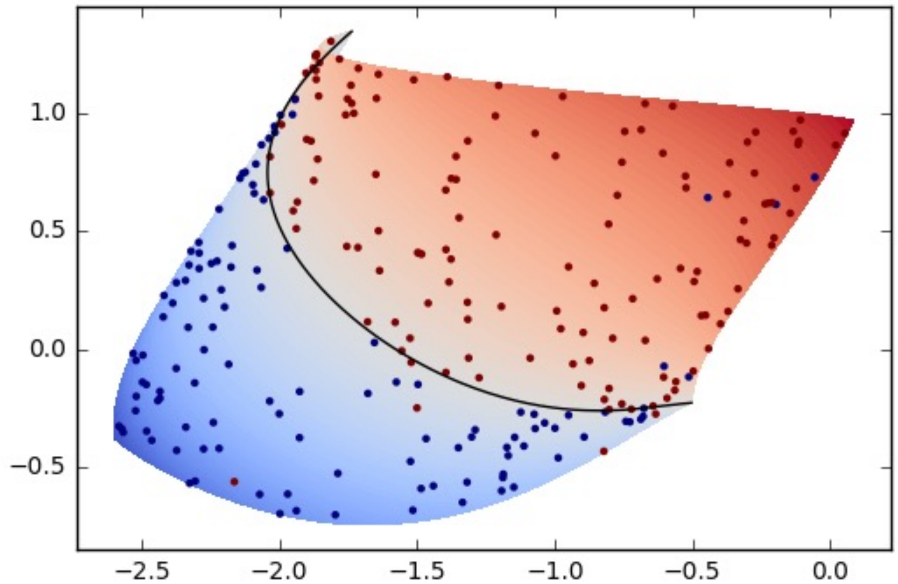
Lecture 12: Neural Networks



Gavin Kerrigan

Spring 2023

Some materials courtesy Padhraic Smyth, Alex Ihler.

Announcements

- HW2 due today at Midnight
  - No late homeworks; lowest score dropped

- In-person lectures resume next Monday (5/1)

- Midterm exam next Friday (5/5)
  - Sample exam in progress (likely available Monday evening)

- Discussion section on Thursday
  - Reviewing sample midterm; midterm review

Review of Logistic Classifiers

Extensions

Neural Networks

The Story So Far

Say we want to predict a binary label y from a feature vector **x**

We are interested in modeling the *conditional probabilities*

$$p(y = 1 \mid \boldsymbol{x})$$

- Recall: why don't we need to model $p(y = 0 \mid \boldsymbol{x})$?

One way to achieve this is using the *logistic classifier* (or *logistic regression)*.

The Story So Far

Logistic Classifiers (for binary classification problems) are models of the form

$$z(\mathbf{x};\boldsymbol{\theta}) = \theta_0 + \sum_{j=1}^{d} \theta_j x_j$$

$$f(\mathbf{x};\boldsymbol{\theta}) = \frac{1}{1 + e^{-z(\mathbf{x};\boldsymbol{\theta})}}$$

- The model output $\mathrm{p_i} = f(\boldsymbol{x_i} \mid \theta)$ can be thought of as the model's belief in $p(y = 1 \mid \boldsymbol{x}_i)$
- Key idea: create an unnormalized "score" z from a weighted linear combination of the features, and turn it into a probability via the sigmoid function

Learned by minimizing the *binary cross-entropy loss*:

$$L_{BCE}(\theta) = \sum_{i=1}^{n} -y_i \log p_i - (1 - y_i)\log(1 - p_i)$$

The Story So Far

Learned by minimizing the *binary cross-entropy loss*:

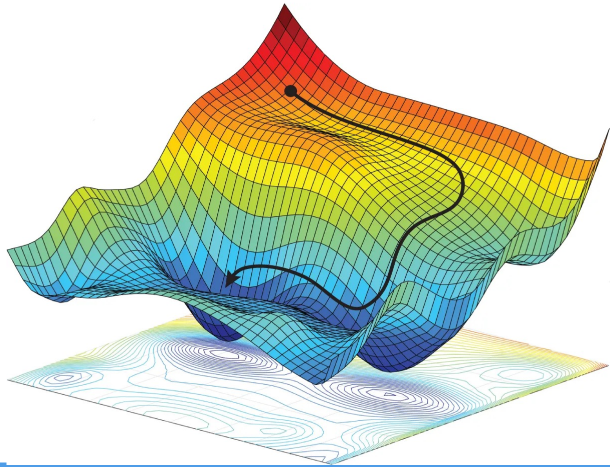
$$L_{BCE}(\theta) = \sum_{i=1}^{n} -y_i \log p_i - (1 - y_i)\log(1 - p_i)$$

- Key idea: we want the predicted probability for the correct class to be as high as possible

We can minimize this function using the gradient descent algorithm

$$\boldsymbol{\theta}^{new} = \boldsymbol{\theta}^{old} - \lambda \cdot \nabla L(\boldsymbol{\theta}^{old})$$

$$\nabla L_{\mathrm{BCE}}(\theta) = \sum_{i=1}^{n} (p_i - y_i)\mathbf{x}_i$$



Review of Logistic Classifiers

Extensions

Neural Networks

Logistic Models with C > 2 classes

How can we use logistic models for problems with more than 2 classes?

Say we have C > 2 classes

Basic idea: define one score function per class, each with their own parameters

$$z_k(\boldsymbol{x} \mid \boldsymbol{\theta}_k) = \theta_{k0} + \sum_{j=1}^{d} \theta_{kj} x_j$$

Score function for *k*th class
k = 1, 2, …, C

Parameters for *k*th score function
$\boldsymbol{\theta}_k = (\theta_{k0}, \theta_{k1}, \dots, \theta_{kd})$

*j*th parameter for *k*th class

- Leads to C (d + 1) parameters in total
- Use $\boldsymbol{\theta}$ to denote the vector of all parameters

Logistic Models with C > 2 classes

Each score function (k = 1, 2, …, C) can be written

$$z_k(\,\boldsymbol{x} \mid \boldsymbol{\theta}_k\,) = \theta_{k0} + \sum_{j=1}^{d} \theta_{kj} x_j$$

To get the predicted class for class k, we apply the *softmax* function

- Generalization of the logistic function to C>2 classes

$$p(\,y = k \mid \boldsymbol{x}, \boldsymbol{\theta}\,) \approx f_k(\,x \mid \theta\,) = \mathrm{softmax}(\mathbf{z}_\mathrm{k}) = \frac{e^{z_k}}{\sum_{\ell=1}^{C} e^{z_\ell}}$$

What is this softmax function doing?

- Takes our C unnormalized scores (the z_k values)
- Exponentiates them so that they are all positive (numerator)
- Normalizes them so that they sum to one (denominator)

The Softmax Function

Let C be the number of classes
and let z_k be the weighted sum going into the k^{th} output

$$f_k(\boldsymbol{x} \mid \boldsymbol{\theta}) = \frac{e^{z_k}}{\sum_{\ell=1}^{C} e^{z_\ell}}$$

Transforms each z_k to be > 0

Normalization constant to ensure numbers sum to 1

Example with C=4 classes:

$z = [z_1 \;\; z_2 \;\; z_3 \;\; z_4] = [2.1 \quad -1.4 \quad 1.2 \quad 0.3]$ ← 4 weighted sums

$e^z = [8.17 \quad 0.25 \quad 3.32 \quad 1.35]$ ← 4 non-negative numbers

$\text{softmax}(z) = [0.62 \quad 0.02 \quad 0.25 \quad 0.10]$ ← 4 probabilities

Notes:

1. Also used in neural networks
2. We have $\sum_{k=1}^{C} f_k(\boldsymbol{x} \mid \boldsymbol{\theta}) = 1$ for every x

Training Multiclass Logistic Models

How do we train a logistic model with C>2 classes?

- Gradient descent!
- Loss function is the *cross-entropy loss*
- We saw the special case of *binary cross-entropy loss* for C=2

The general form of the cross entropy loss is:

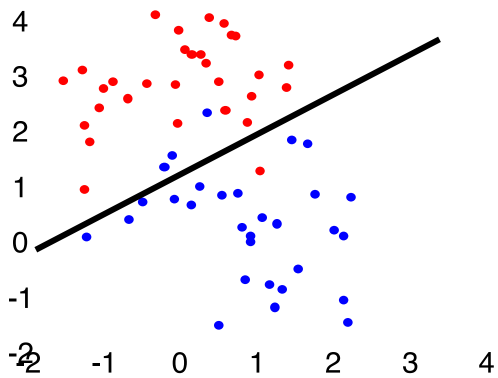
$$L_{CE}(\theta) = \sum_{i=1}^{n} -\log f_{y_i}(\, \boldsymbol{x_i} \mid \boldsymbol{\theta}\,)$$

Probability predicted by the model for the correct class y_i for the training datapoint $\mathbf{x}_i$

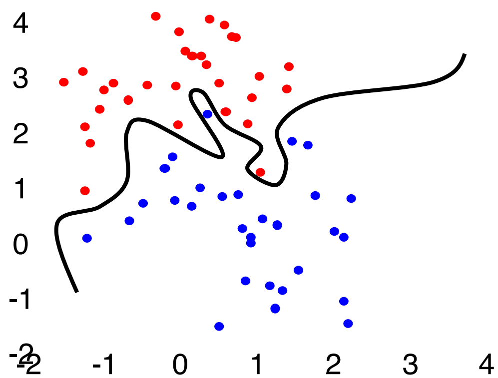
Non-Linear Boundaries with Logistic Models?

Adding Non-Linearity to a Logistic Model

A logistic model has
linear decision boundaries



Can we use a logistic model
to get non-linear ("curvy")
decision boundaries"?



Adding Non-Linearity to a Logistic Model

One way to do this is to add additional higher order features to the model

e.g., add features such as powers like $(x_1)^2$, $(x_2)^2$, the product $x_1\ x_2$, and so on

- We saw *polynomial feature expansions* in the context of regression

This is useful: we can still train the logistic model in the same way
(these are just extra ”pre-computed” features to gradient descent)

.....and in the original space (x_1, x_2, ...) we will get non-linear
(e.g., quadratic) boundaries

However, there are a few problems with this approach.......

- How do we choose which features to add?
- More features leads to slower training/predictions, more memory usage

Summary of Key Concepts in Logistic Models

- A logistic model has 2 parts
  - A linear weighted sum
  - A sigmoid that maps the weighted sum to the interval [0, 1]

- We can interpret the outputs of the logistic model as class probabilities

- We can generalize from 2 classes to K classes by having a set of weights for each class

- We can learn logistic models using the log-loss function and using gradient descent for optimization

Strengths and Weaknesses of Logistic Classifiers

- Strengths
  - Easy to fit (simple gradient, convex loss function)
  - Easy to interpret (one weight per feature)

- Weaknesses
  - Linear decision boundary: might not be good enough for more complex classification problems

Review of Logistic Classifiers

Extensions

Neural Networks

A Little History on Neural Networks

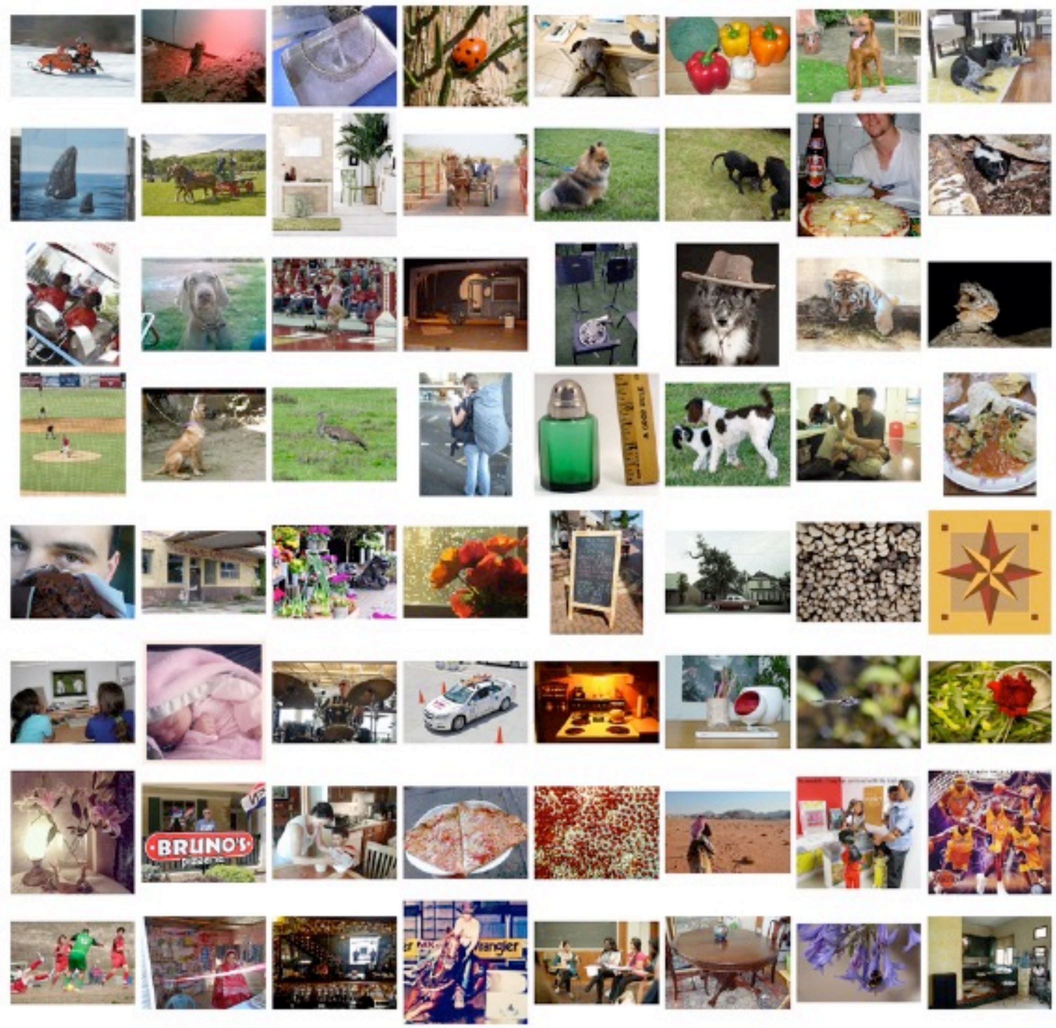
- Phase 1, 1950s to 1970s
  - Logistic-like models, no hidden units
  - Initial enthusiasm died out
- Phase 2, 1980s to 2000s
  - Invention of backpropagation: could train models with hidden units
  - But training was slow, data was scarce….initial enthusiasm died out
- Phase 3, 2010s to present
  - Demonstrations of the power of deep learning models
  - (re)invention of a technique called stochastic gradient
  - Commercial successes, great enthusiasm….

ImageNet

A testbed for evaluating
image classification algorithms

Over 10 million images

1000 class labels



From Russakovsky et al, ImageNet Large Scale Visual Recognition Challenge, 2015

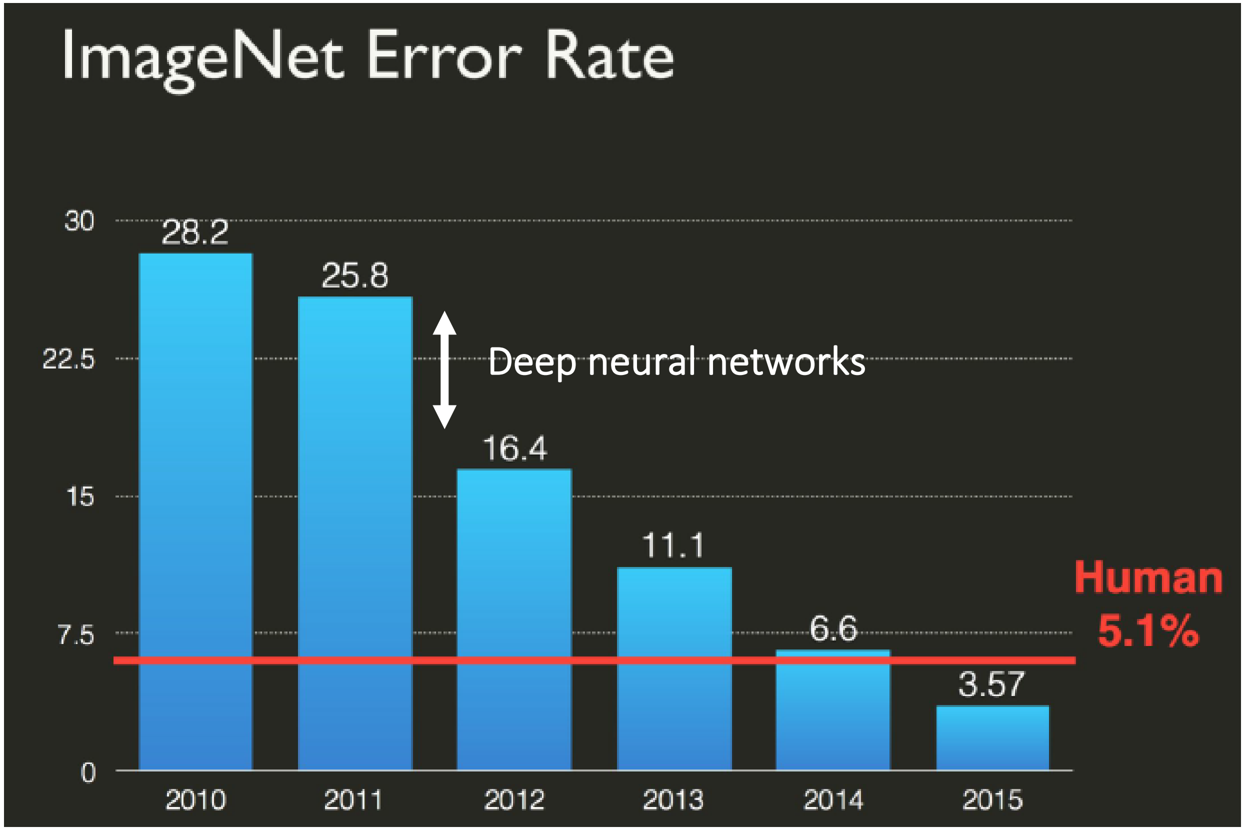
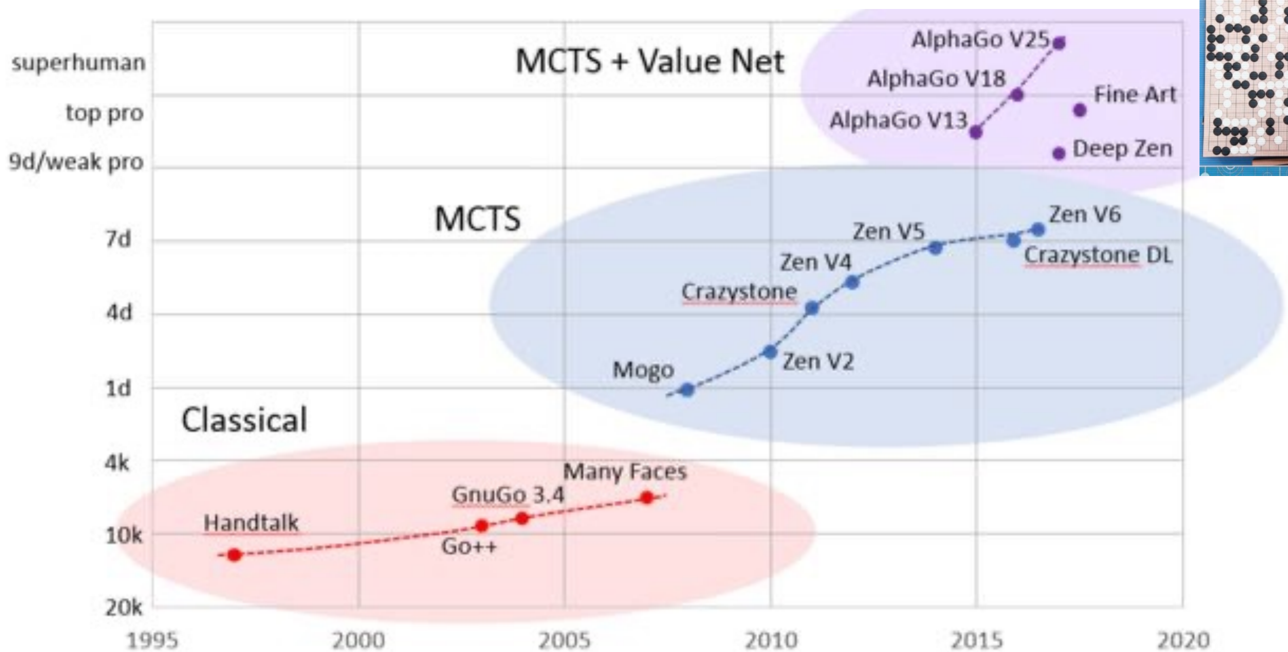
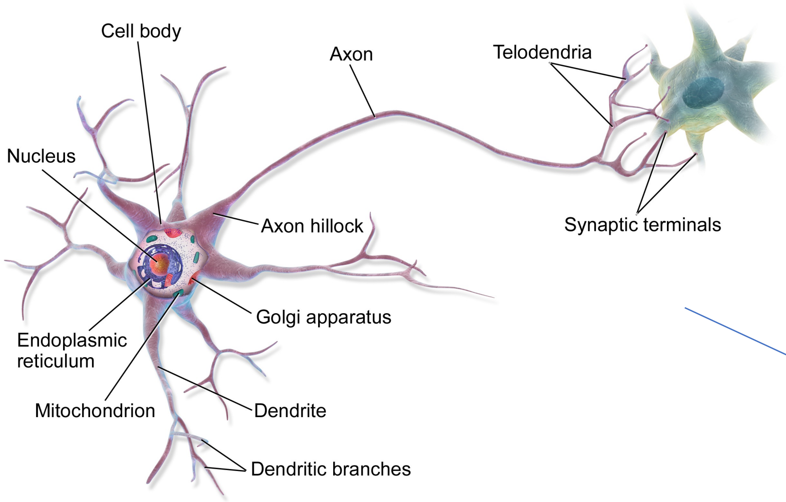


Figure from Kevin Murphy, Google



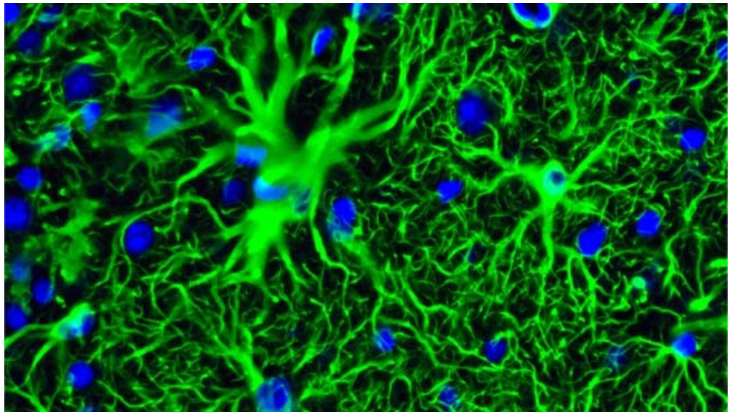
https://www.reddit.com/r/baduk/comments/6ttyyz/better_graph_of_go_ai_strength_over_time/

This is in your brain



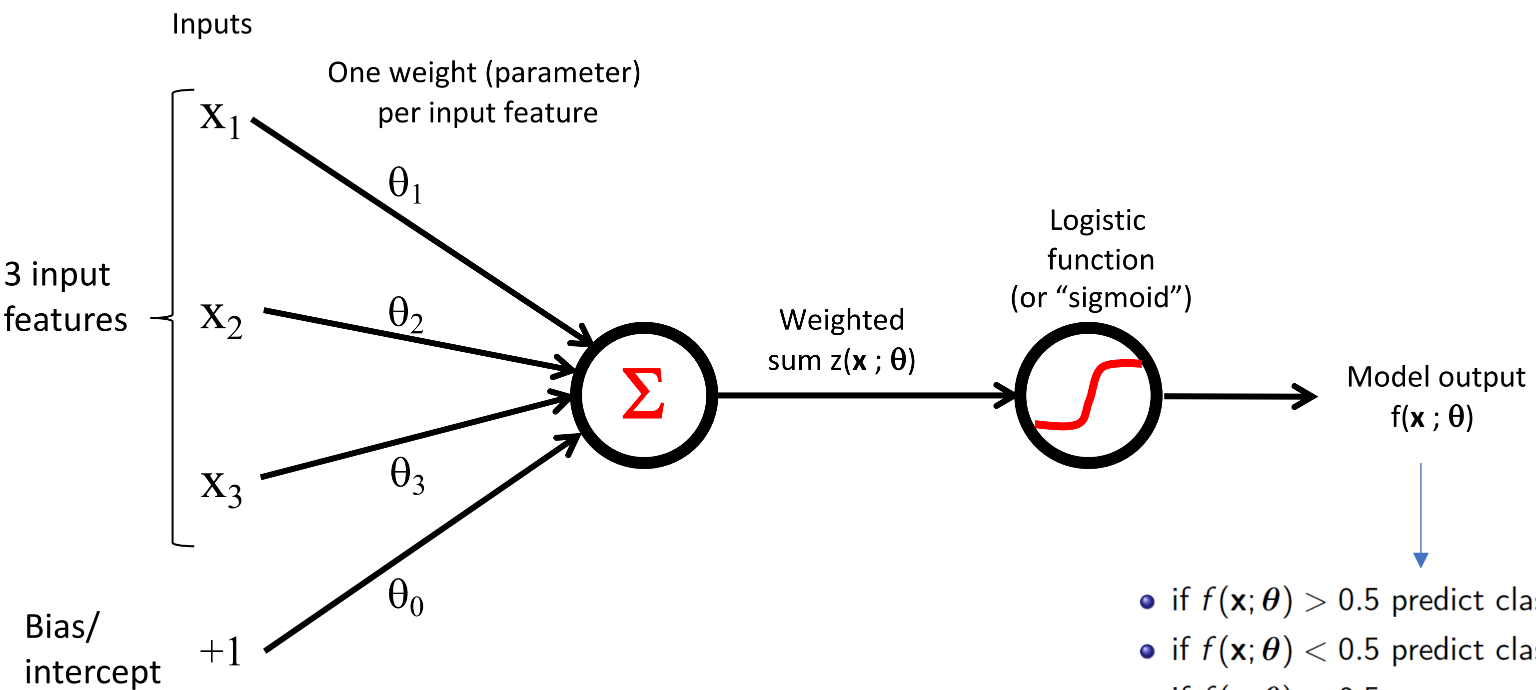
By BruceBlaus - Own work, CC BY 3.0,
https://commons.wikimedia.org/w/index.php?curid=28761830

Neuronal circuits



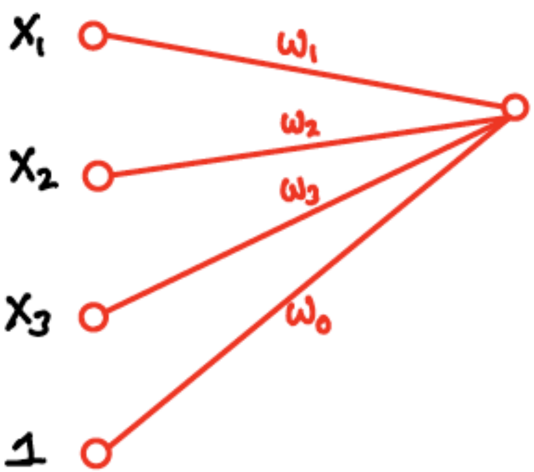
From https://neuronline.sfn.org/scientific-research/circuit-mapping-networks

Block Diagram of a Logistic Classifier



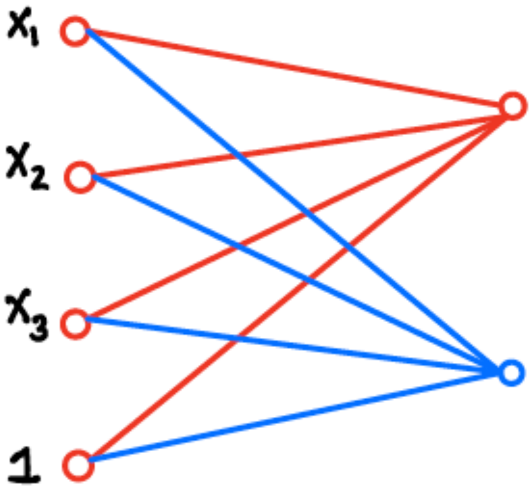
- if $f(\mathbf{x};\boldsymbol{\theta}) > 0.5$ predict class label 1
- if $f(\mathbf{x};\boldsymbol{\theta}) < 0.5$ predict class label 0
- if $f(\mathbf{x};\boldsymbol{\theta}) = 0.5$ we are on the decision boundary

The Sigmoid as a Computational Unit



$$h(x) = \frac{1}{1 + e^{-w_0 - \sum w_j x_j}}$$

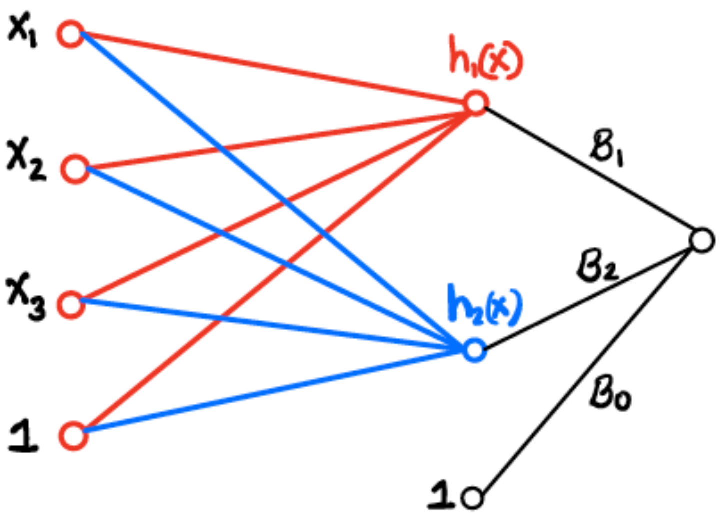
Two Sigmoids



$$h_1(x) = \frac{1}{1 + e^{-w_0 - \Sigma w_j x_j}}$$

$$h_2(x) = \frac{1}{1 + e^{-w_0 - \Sigma w_j x_j}}$$

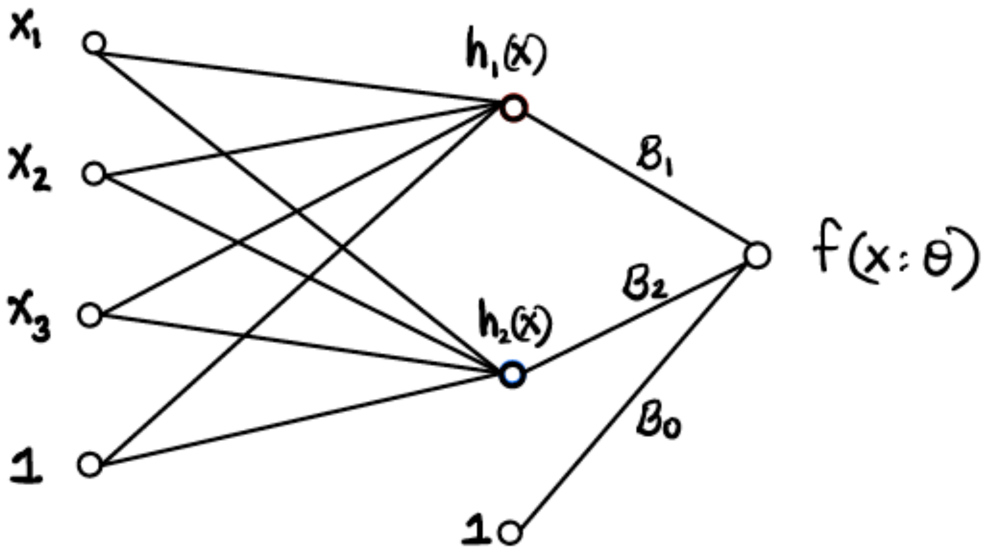
A Simple Neural Network



$$f(x;\theta) = \frac{1}{1 + e^{-\beta_0 - \beta_1 h_1(x) - \beta_2 h_2(x)}}$$

Terminology: Hidden Units

Hidden Units



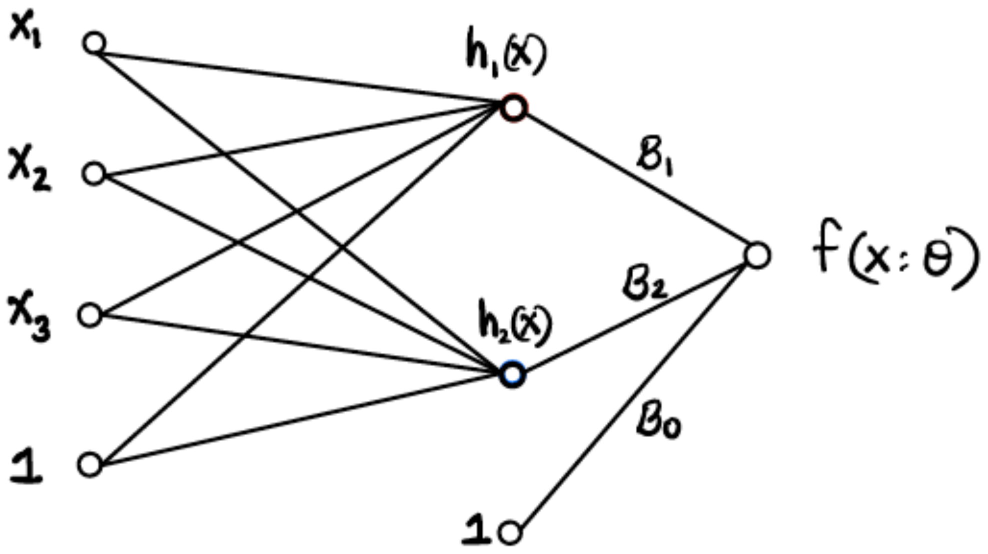
Hidden Units = non-linear functions of inputs

Here the non-linearity is the sigmoid (but other options are often used)

The non-linear function can also be referred to as an “activation function”

Terminology: Layers

Hidden Layer

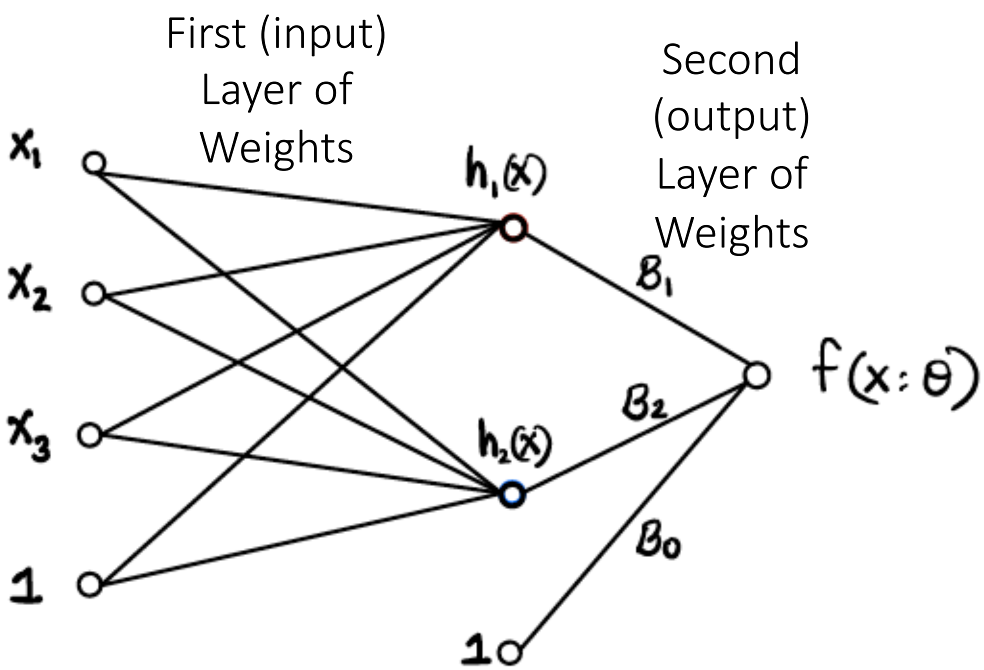


We think of neural networks in terms of "layers"

This is a "single hidden layer" neural network

Later we will see networks with multiple hidden layers

Terminology: Weight Layers

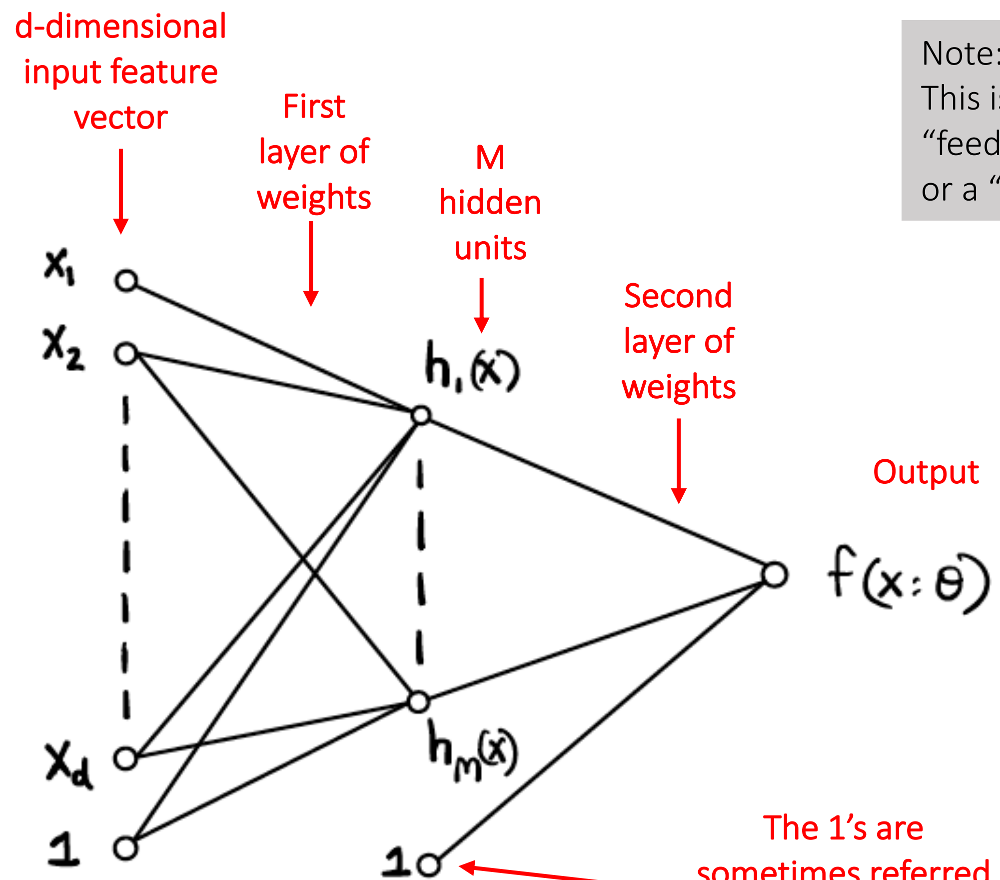


We can also refer to layers of weights

Here we have 2 layers:

1. input to hidden units
2. hidden units to output

A General Single Hidden Layer Neural Network

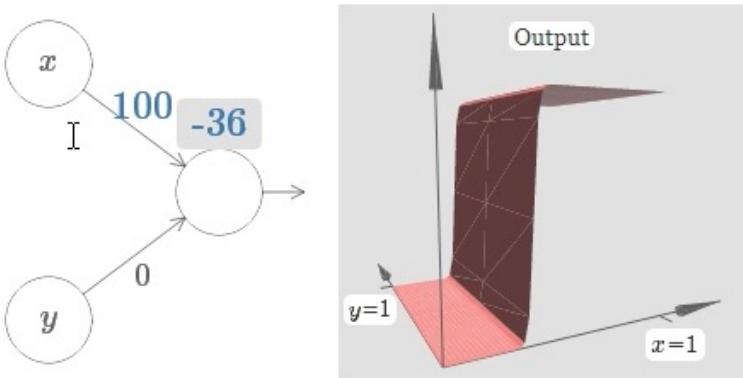


Note:
This is also sometimes referred to as "feedforward" neural network
or a "multilayer perceptron"

Parameters θ = all parameters from all layers

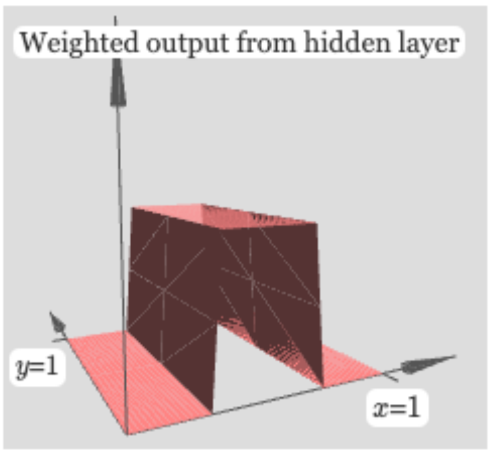
How many for a neural network with M hidden units? $(d+1)*M + M+1 = O(dM)$

What can a Neural Network Represent?



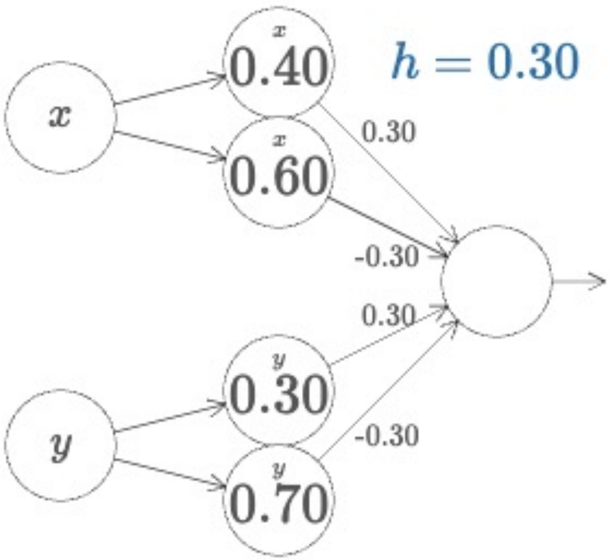
Single logistic
with no weight on
the 2nd feature (y)

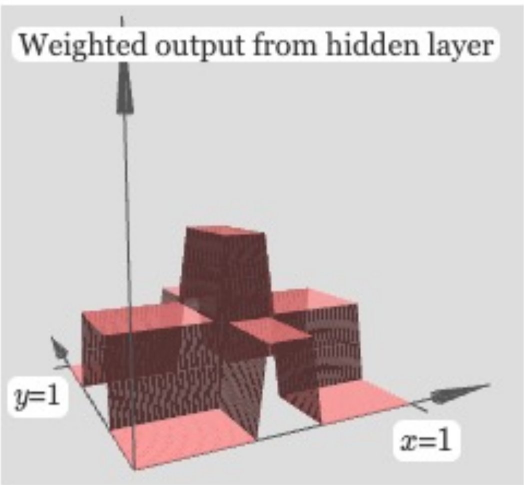
Combined output of
2 logistic functions
(hidden units)
in a neural network



Visualizations from http://neuralnetworksanddeeplearning.com/chap4.html

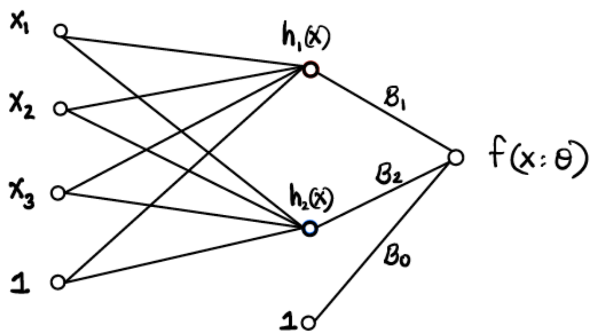
What can a Neural Network Represent?

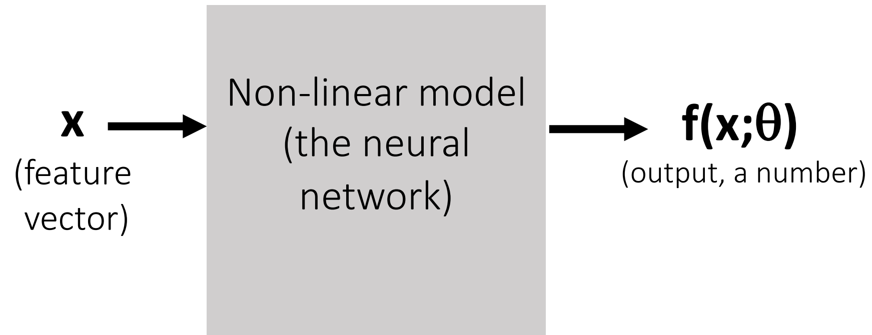




In theory, as number of hidden units goes to infinity,
a neural network with a single hidden layer can represent any smooth function

A Neural Network as a Classifier





A neural network is a non-linear mapping from a feature vector to an output

If the outputs lie between 0 and 1 (e.g., via a sigmoid) then we can interpret the output as f(**x** ; **θ**) = P(y = 1 | **x**), i.e., as class probabilities

…some small details remain….e.g., how to train the neural network!

Neural Networks have Non-Linear Decision Boundaries

..and, adding more hidden units results in more complex decision boundaries

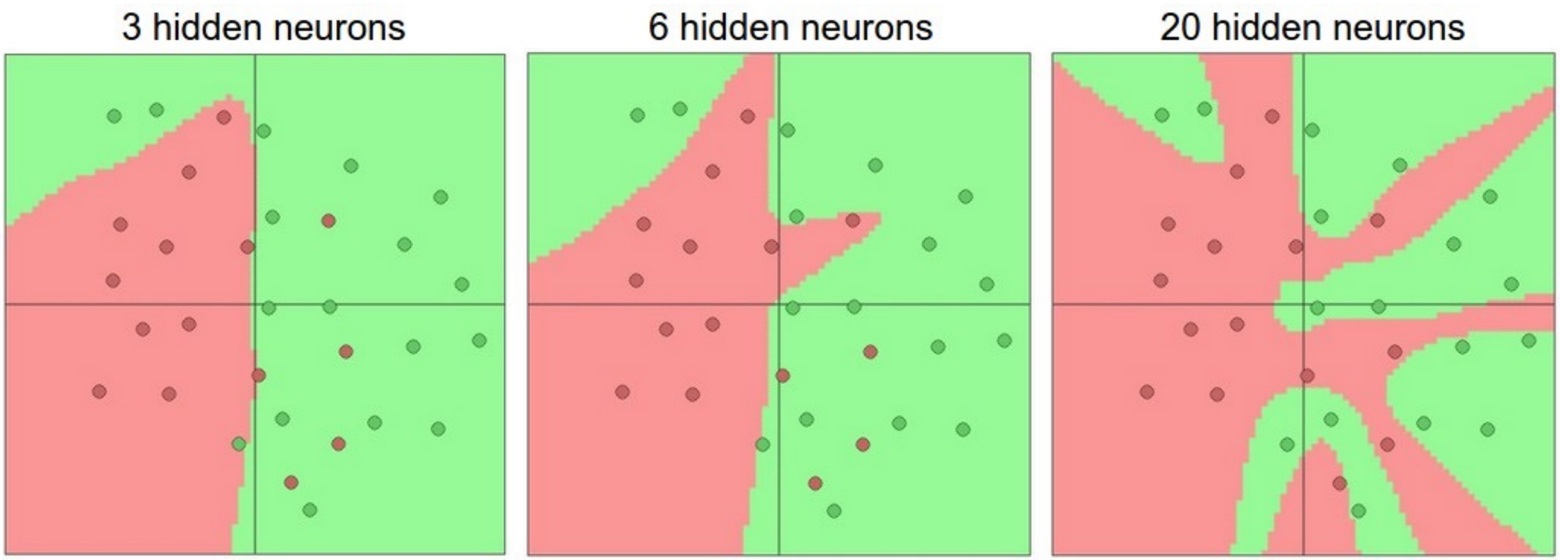


Figure from https://cs231n.github.io/neural-networks-1/

Math Notation for a Single Hidden Layer NN

Let $g(z)$ represent a general activation function, e.g., in our examples

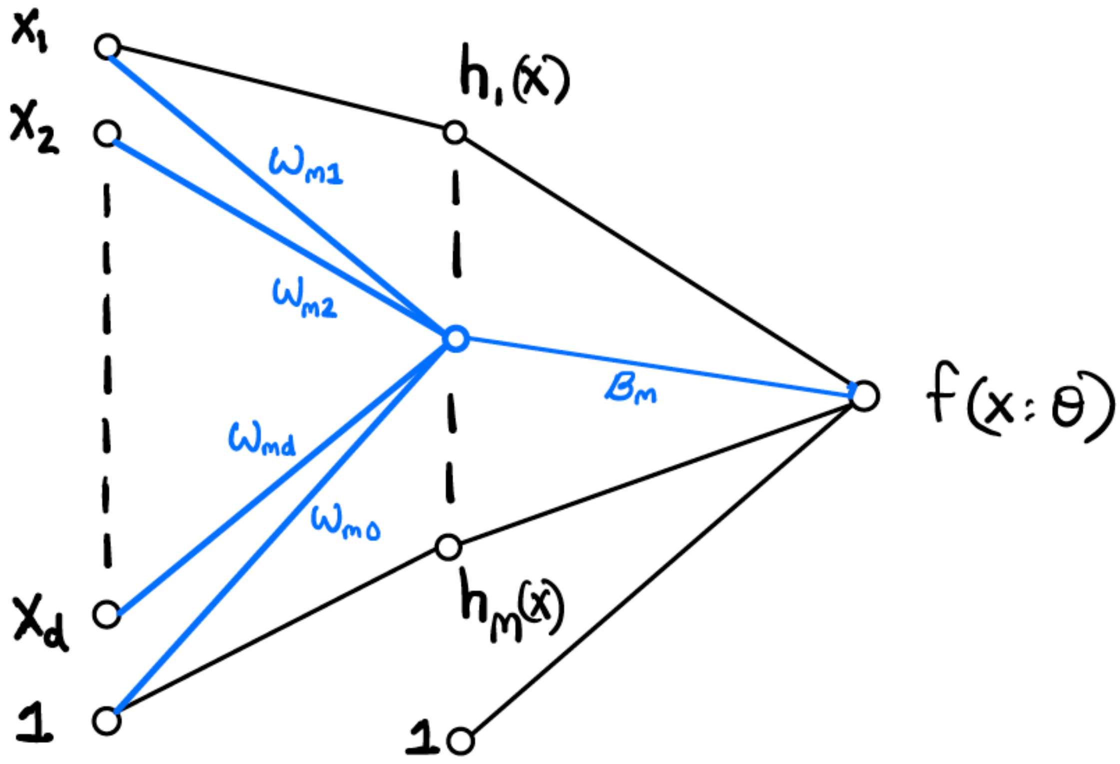
$$g(z) = \frac{1}{1+e^{-z}} = \text{sigmoid function}$$

Let $h_m(\mathbf{x})$ be the output of the mth hidden unit:

$$h_m(\mathbf{x}) = g\left(w_{m0} + \sum_{j=1}^{d} w_{mj} \cdot x_j\right)$$

where w_{mj} is the weight from input x_j to hidden unit h_m, with $j = 0, \ldots, d$

Math Notation for a Single Hidden Layer NN



Math Notation for a Single Hidden Layer NN

The output of the network is

$$
\begin{aligned}
f(\mathbf{x}; \boldsymbol{\theta}) &= g\left(\beta_0 + \sum_{m=1}^{M} \beta_m \cdot h_m(\mathbf{x})\right) \\
&= g\left(\beta_0 + \sum_{m=1}^{M} \beta_m \cdot g\left(w_{m0} + \sum_{j=1}^{d} w_{mj} \cdot x_j\right)\right)
\end{aligned}
$$

The parameters $\boldsymbol{\theta}$ of the neural network are

$$\boldsymbol{\theta} = (\mathbf{w}_1, \ldots, \mathbf{w}_M, \boldsymbol{\beta})$$

where $\mathbf{w}_m$ are the set of weights going from the input to the mth hidden unit and $\boldsymbol{\beta}$ is the set of M weights going from hidden units to the output.

Weight Matrix Notation

$$W = \begin{bmatrix} \cdots\cdots w_1 \cdots\cdots \\ w_2 \\ \vdots \\ \cdots\cdots w_M \cdots\cdots \end{bmatrix} = \overbrace{\begin{bmatrix} w_{10} & w_{11} & \cdots\cdots & w_{1d} \\ \vdots & & & \vdots \\ w_{M0} & w_{M1} & \cdots\cdots & w_{Md} \end{bmatrix}}^{d+1} \Big\updownarrow M \qquad x = \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{pmatrix}$$

Weight matrix
for the first layer

Weight Matrix Notation

$$W = \begin{bmatrix} \cdots\; w_1 \;\cdots \\ w_2 \\ \vdots \\ \cdots\; w_M \;\cdots \end{bmatrix} = \overbrace{\begin{bmatrix} w_{10} & w_{11} & \cdots & w_{1d} \\ \vdots & & & \vdots \\ w_{M0} & w_{M1} & \cdots & w_{Md} \end{bmatrix}}^{d+1} \updownarrow M \qquad \mathbf{x} = \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_d \end{pmatrix}$$

Weight matrix for the first layer

$$W\,\mathbf{x} = \begin{pmatrix} w_1 \mathbf{x} \\ w_2 \mathbf{x} \\ \vdots \\ w_M \mathbf{x} \end{pmatrix} \updownarrow M$$

M rows x (d+1) columns

(d+1) x 1 column vector

vector of M weighted sums (one per hidden unit)

$$g(W x) = g\begin{pmatrix} w_1 x \\ w_2 x \\ \vdots \\ w_M x \end{pmatrix} = \begin{pmatrix} g(w_1 x) \\ g(w_2 x) \\ \vdots \\ g(w_M x) \end{pmatrix} = \begin{pmatrix} h_1(x) \\ h_2(x) \\ \vdots \\ h_m(x) \end{pmatrix}$$

Compute the M hidden unit responses,
applying the non-linearity g( ) element wise to the **W x** vector

$$\beta = (\beta_0, \beta_1, \beta_2, \cdots \beta_M)$$

Write the 2nd layer of weights
as a vector

$$h(x) = \begin{pmatrix} 1 \\ h_1(x) \\ h_2(x) \\ \vdots \\ h_m(x) \end{pmatrix}$$

Augment the hidden
unit responses with a
"1" for the bias term

Element-wise
application of non-linear
g( ) activation function

$$f(\mathbf{x};\theta) = g(\ \beta\ \mathbf{h}(\mathbf{x})) = g(\ \beta\ g(\mathbf{W}\ \mathbf{x}))$$

Matrix-vector representation of a neural network with a single hidden layer

What we have learned so far….

We can build simple neural networks by combining logistic functions in a hierarchical manner

Hidden units compute functions of the input, and the hidden unit values are then combined to produce an output

The hidden units use a non-linear function (e.g., a sigmoid), also known as an activation function

A neural network can be used to generate complex output functions, which in turn produce complex decision boundaries

Many unanswered questions remain....

- How can we train (learn the weights) a neural network?
- Can we have more than 1 output?
- Can we have more than 1 hidden layer?
- Can we use other non-linearities besides sigmoid?
- How should we select the “architecture” of a neural network?
- Can we use neural networks with pixels, with text, etc?

Wrapup

- Logistic models
  - Can be extended to C>2 classes
  - Feature expansions allow for non-linear decision boundaries
    - But how do we pick the features?

- Neural networks
  - Extension of logistic models
  - Allows for learning new feature representations automatically