Logistic regression and simple multi-layer neural networks

Roger Levy

9.19: Computational Psycholinguistics

27 October 2021

Agenda for the day

- Review logistic regression (case study: *binomial ordering preferences*)
- Limitations of linear classifiers like logistic regression
- Basic multi-layer neural networks & backpropagation
- Expressing and learning solutions to non-linear classification problems
- Vanishing gradients and activation functions

Recap: binomial ordering preferences

- In each pair, which phrase sounds more natural?

pepper and salt	*salt and pepper*
hit and run	*run and hit*
gold and silver	*silver and gold*
deer and trees	*trees and deer*
drink and food	*food and drink*
skirts and sweaters	*sweaters and skirts*
bishops and seamstresses	*seamstresses and bishops*
few and unfavorable	*unfavorable and few*
cat and mouse	*mouse and cat*
quilting and sewing	*sewing and quilting*
interest and principal	*principal and interest*

Multiple, cross-cutting constraints

$\{X_i\}$

Constraint	Example	Strength
Iconic/scalar sequencing	*open and read*	**20**
Perceptual markedness	*deer and trees*	**1.7**
Formal markedness	*change and improve*	**1.4**
Power	*food and drink*	**1**
Avoid final stress	*confuse and disorient*	**0.5**
Short<Long	*cruel and unusual*	**0.4**
Frequent<Infrequent	*neatly and sweetly*	**0.3**

$\{\beta_i\}$

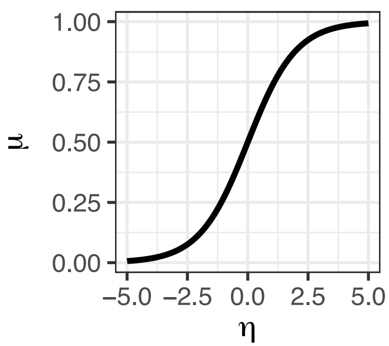
- **Logistic regression** to capture effects on ordering preference:

$$\underline{\eta} = \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_N X_N$$

"goodness score"

$$P(\text{“success”}) = \underline{\frac{e^\eta}{1+e^\eta}}$$

a.k.a. **mean** μ



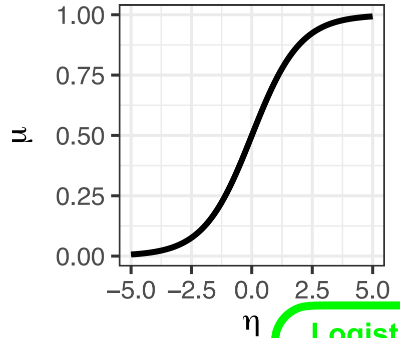
Logistic (sigmoid) activation function

A two-constraint example

- Constraints: word **length** (# syllables) and word **frequency**

$$\eta = \beta_{\textsf{Syl}} X_{Syl} + \beta_{\textsf{Freq}} X_{Freq}$$

$$P(\text{“success”}) = \frac{e^{\eta}}{1 + e^{\eta}}$$



Logistic (sigmoid) activation function

Arbitrarily define:
"success"↔alphabetical ordering

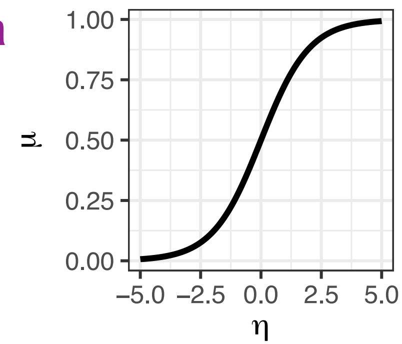
	Short≺Long?	Freq≺Infreq?
calm and relaxed	✔	✔
big and thick	***n/a***	✔
down and out	***n/a***	✘
cruel and unusual	✔	✘
anger and spite	✘	✔
crochet and knit	✘	✘

Learning constraint weights

Goal: Estimate good values from data

$$\eta = \boxed{\beta_{\text{Syl}}} X_{Syl} + \boxed{\beta_{\text{Freq}}} X_{Freq}$$

$$P(\text{“success”}) = \frac{e^{\eta}}{1 + e^{\eta}}$$



Logistic (sigmoid) activation function

	Short<Long?	X_{Syl}	Freq<Infreq	X_{Freq}
calm and relaxed	✔	1	✔	1
big and thick	*n/a*	0	✔	1
down and out	*n/a*	0	✘	-1
cruel and unusual	✔	1	✘	-1
anger and spite	✘	-1	✔	1
crochet and knit	✘	-1	✘	-1

Then, e.g. find maximum-likelihood estimates $\langle \hat{\beta}_{\text{Syl}}, \hat{\beta}_{\text{Freq}} \rangle$

Maximum of the likelihood surface

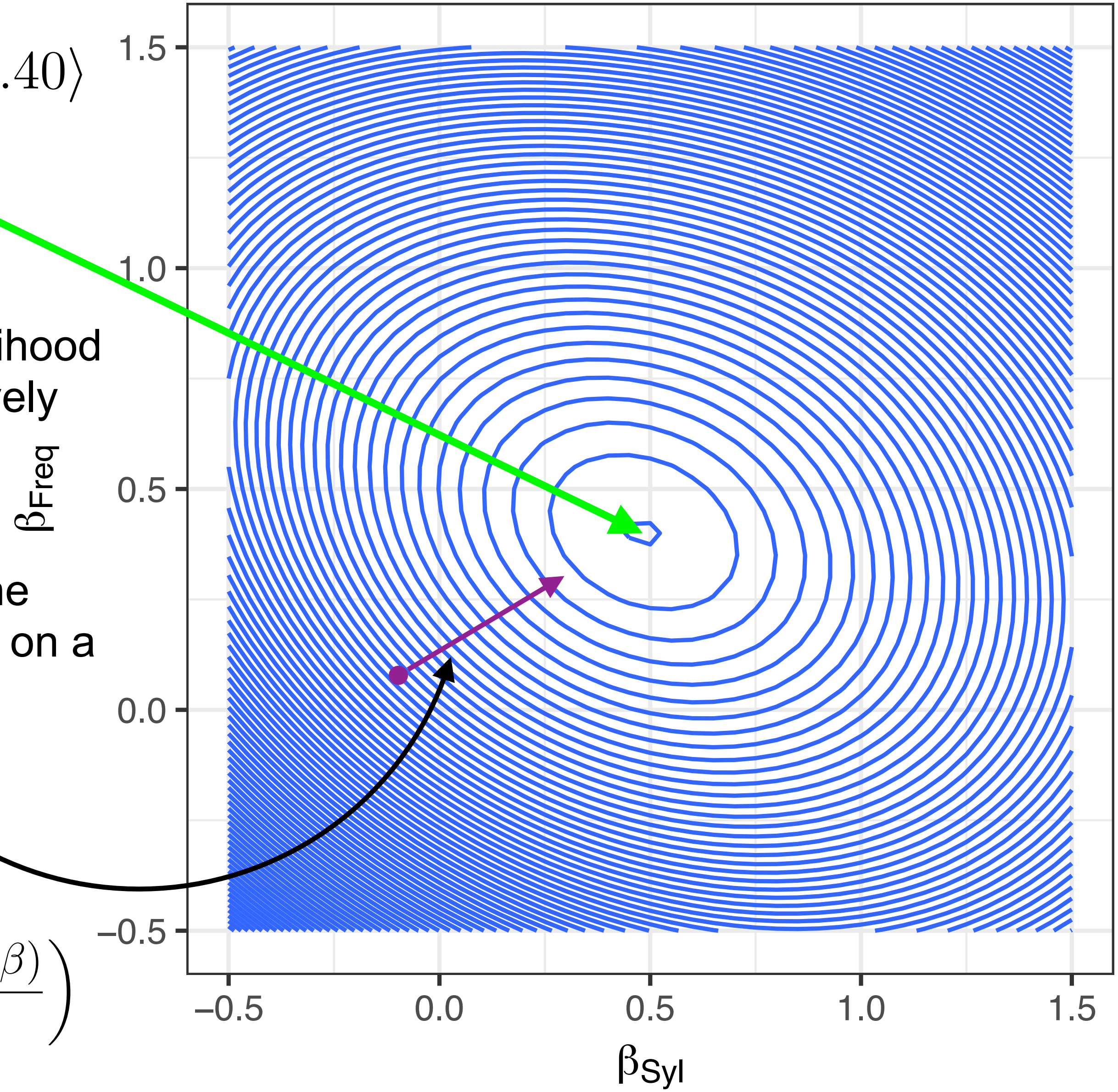
$\langle\widehat{\beta}_{Syl},\widehat{\beta}_{Freq}\rangle = \langle 0.48, 0.40\rangle$

For logistic regression, likelihood surface is **convex** — relatively easy to find optimum

Crucial notion: ***gradient***, the "derivative in all directions" on a multidimensional surface

$$\nabla_\beta \mathsf{Lik}(\mathsf{Data};\beta)$$

$$\left(\frac{\partial\,\mathsf{Lik}(\mathsf{Data};\beta)}{\partial\beta_1}, \frac{\partial\,\mathsf{Lik}(\mathsf{Data};\beta)}{\partial\beta_2}\right)$$



Limitations of logistic regression

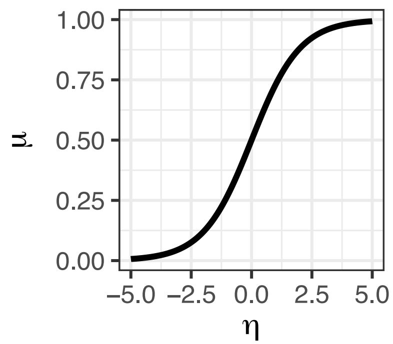
- Logistic regression defines a ***hyperplane*** boundary separating $P(\text{"success"} \mid X) > 0.5$ from $P(\text{"success"} \mid X) < 0.5$

$$\langle \widehat{\beta}_{Syl}, \widehat{\beta}_{Freq} \rangle = \langle 0.48, 0.40 \rangle$$

$$\eta = 0.48 X_{Syl} + 0.4 X_{Freq}$$

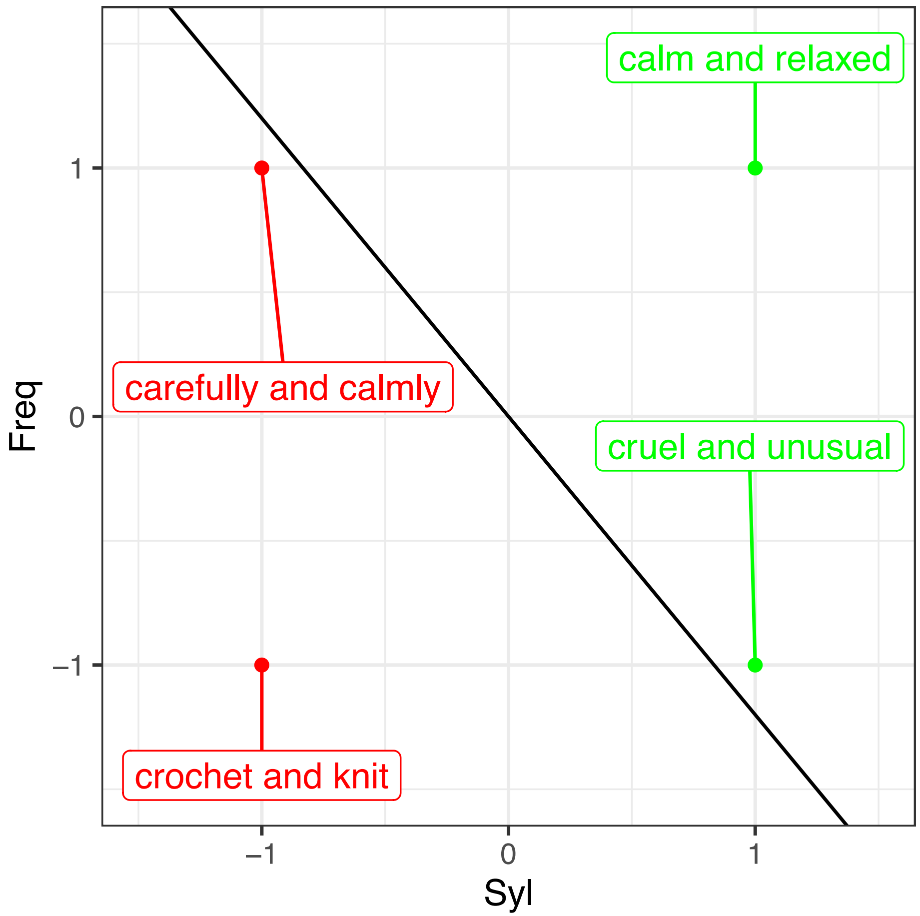
$$P(\text{“success”}) = \frac{e^{\eta}}{1 + e^{\eta}}$$

Logistic (sigmoid) activation function



$$0 = 0.48 X_{Syl} + 0.4 X_{Freq}$$

$$X_{Freq} = -\frac{0.48}{0.4} X_{Syl}$$

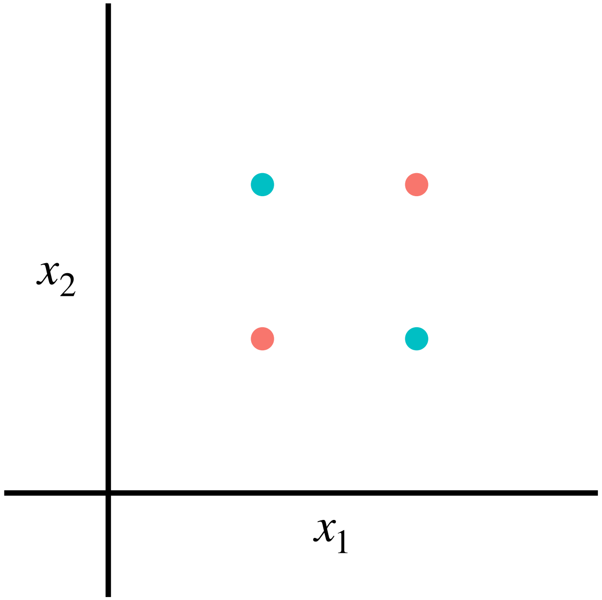


Problems that aren't linearly separable

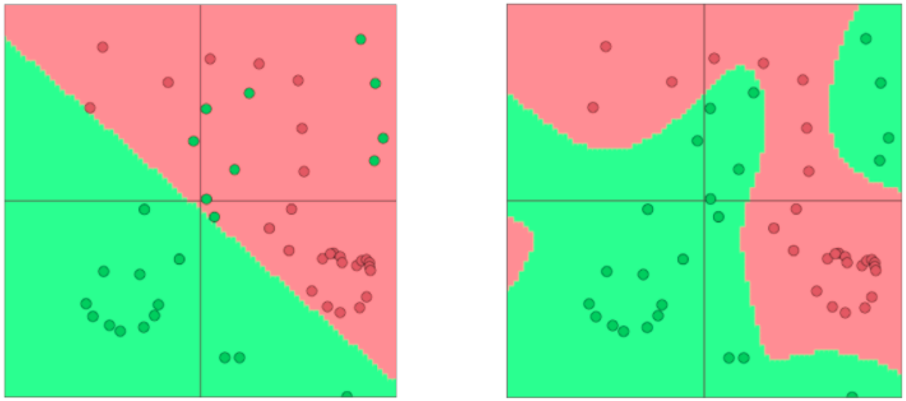
- But many prediction problems aren't linearly separable

XOR problem

x_1	x_2	Class
0	0	1
0	1	0
1	0	0
1	1	1

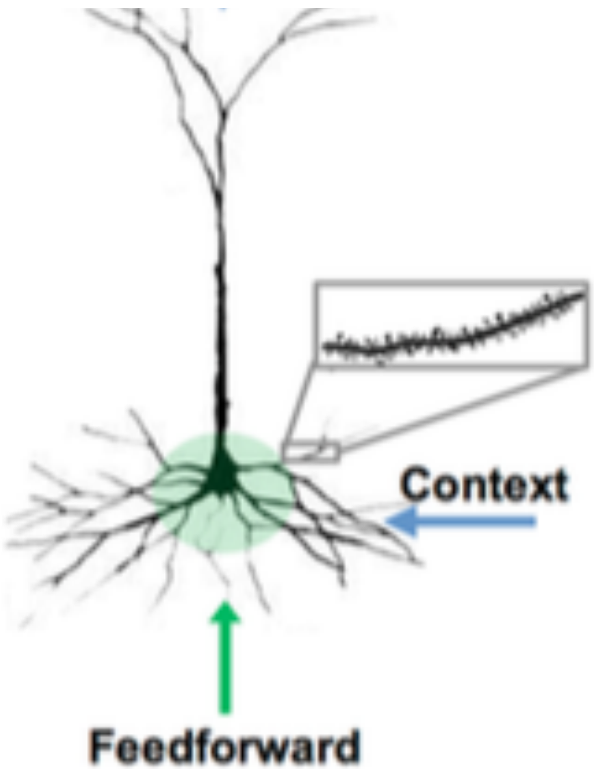


More generally, we want flexibly-shaped class boundaries:

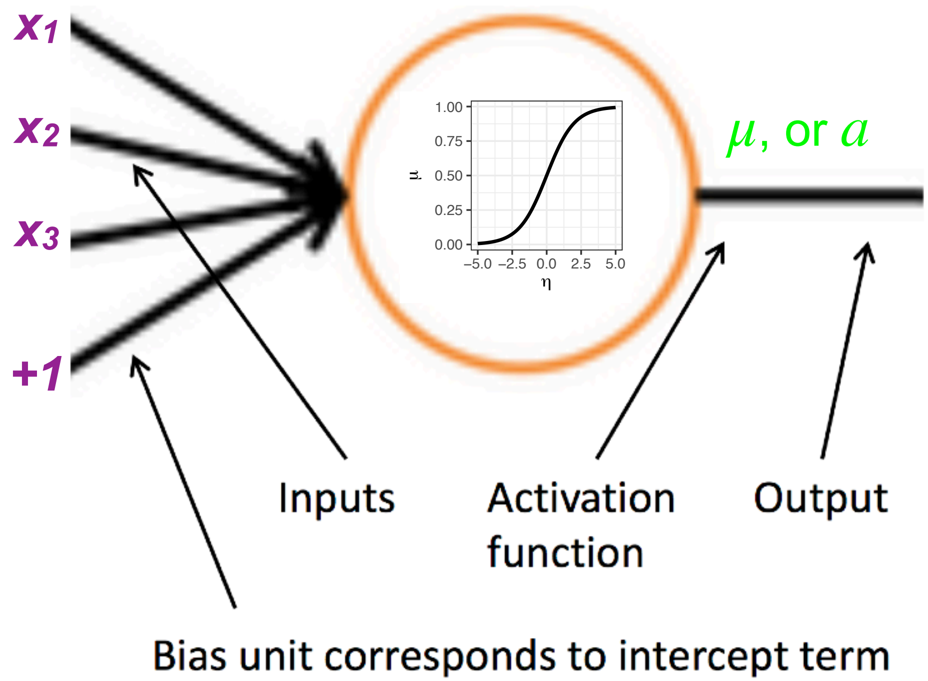


Logistic regression as a “neuron”

Biological neuron



Artificial neuron



$$\eta = \sum_i \beta_i X_i \qquad \mu = \frac{e^\eta}{1 + e^\eta}$$

$$\downarrow \qquad\qquad \downarrow$$

$$z = Wx + b \qquad a = f(z)$$

Neurons are organized in networks!

BULLETIN OF
MATHEMATICAL BIOPHYSICS
VOLUME 5, 1943

A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY

WARREN S. MCCULLOCH AND WALTER PITTS

FROM THE UNIVERSITY OF ILLINOIS, COLLEGE OF MEDICINE,
DEPARTMENT OF PSYCHIATRY AT THE ILLINOIS NEUROPSYCHIATRIC INSTITUTE,
AND THE UNIVERSITY OF CHICAGO

Because of the "all-or-none" character of nervous activity, neural events and the relations among them can be treated by means of propositional logic. It is found that the behavior of every net can be described in these terms, with the addition of more complicated logical means for nets containing circles; and that for any logical expression satisfying certain conditions, one can find a net behaving in the fashion it describes. It is shown that many particular choices among possible neurophysiological assumptions are equivalent, in the sense that for every net behaving under one assumption, there exists another net which behaves under the other and gives the same results, although perhaps not in the same time. Various applications of the calculus are discussed.

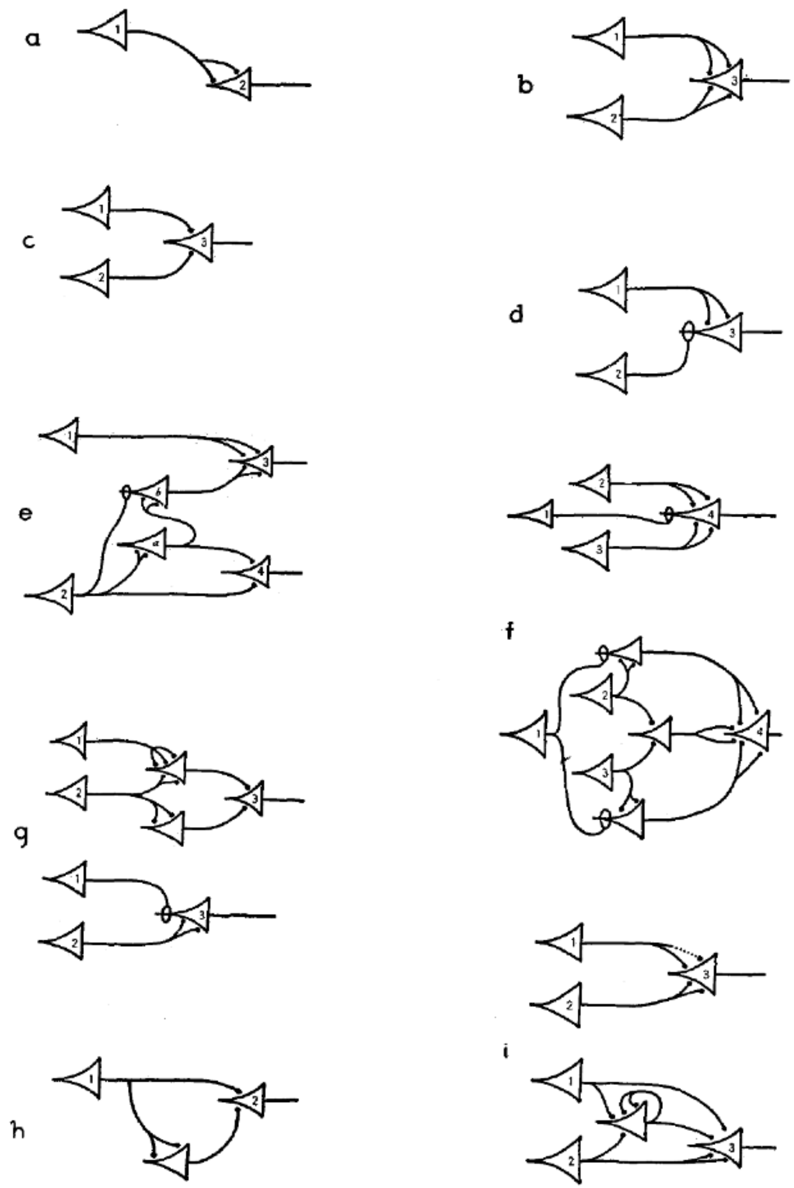
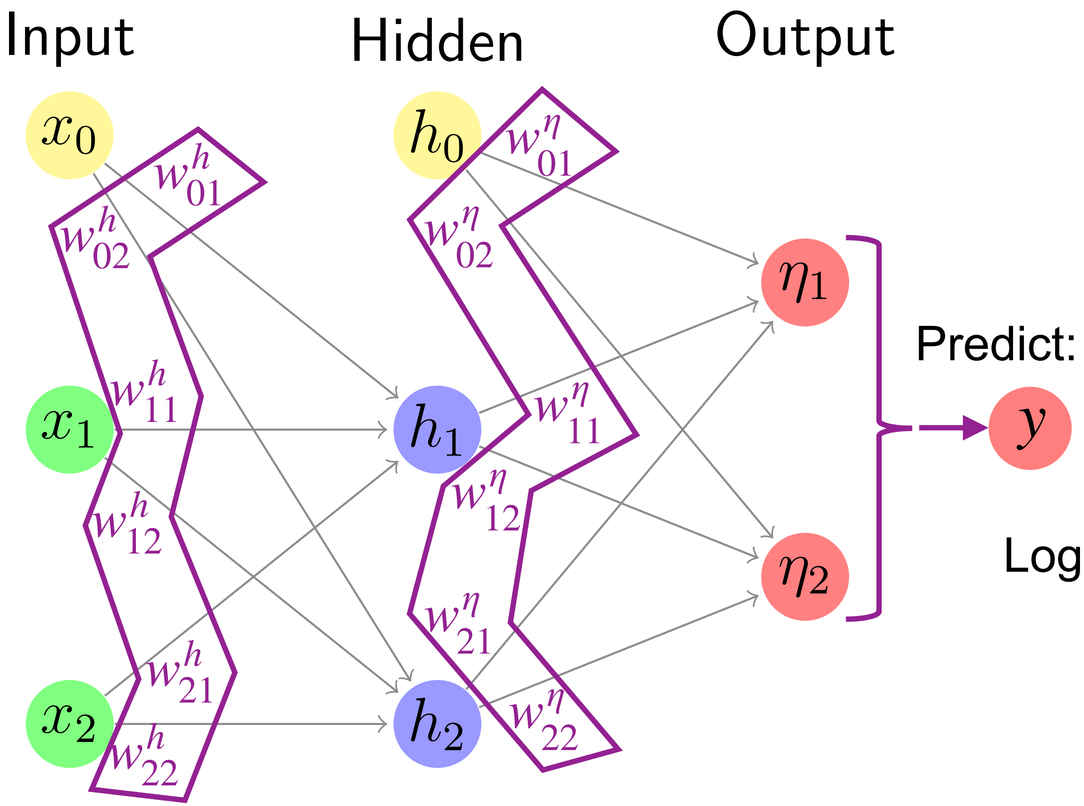


FIGURE 1

A simple single-hidden-layer neural network



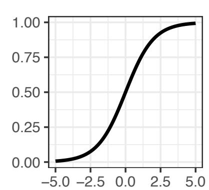
Predict: is y class 1 or class 2?

Logistic ("softmax") output prediction:

$$P(y = 1 \mid x_1, x_2) = \frac{e^{\eta_1}}{e^{\eta_1} + e^{\eta_2}}$$

$$\mathbf{H} = \mathbf{W}^{\mathbf{h}}\mathbf{X} \qquad \eta = \mathbf{W}^{\eta} f(\mathbf{H})$$

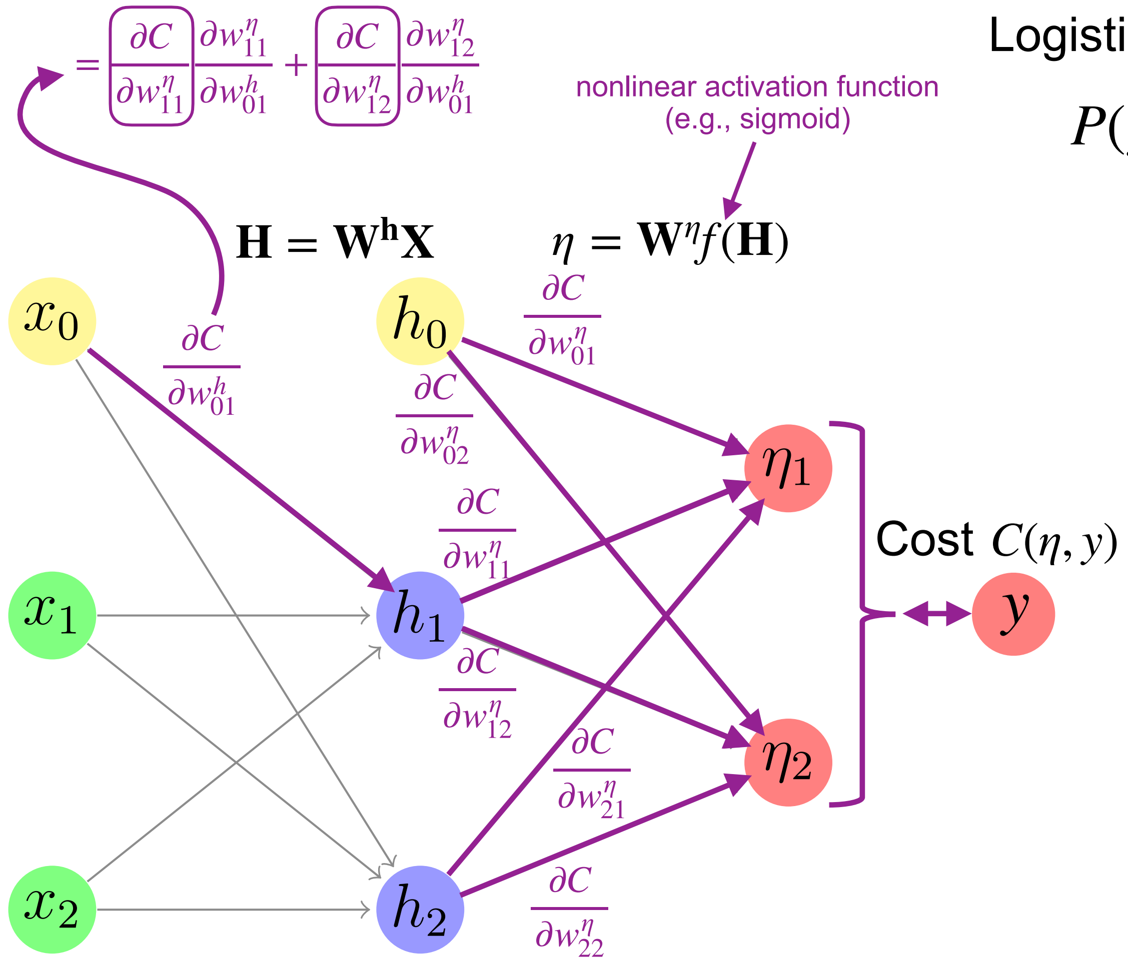
nonlinear activation function
(e.g., sigmoid)



$$P(\mathbf{y} \mid \mathbf{X}) = \text{softmax}(\eta)$$

$$P(\mathbf{y} \mid \mathbf{X}) = \text{softmax}(\mathbf{W}^{\eta} f(\mathbf{W}^{\mathbf{h}}\mathbf{X}))$$

Gradient descent with neural networks



Logistic ("softmax") output prediction:

$$P(y = 1 \mid x_1, x_2) = \frac{e^{\eta_1}}{e^{\eta_1} + e^{\eta_2}}$$

To improve the model's weights, we iteratively compute $\nabla_{\mathbf{w}} C(\eta, y)$ and move the weights in that direction

Chain rule of calculus:

if $y = f(u_1, \ldots, u_n)$ and $u_i = g_i(x)$,

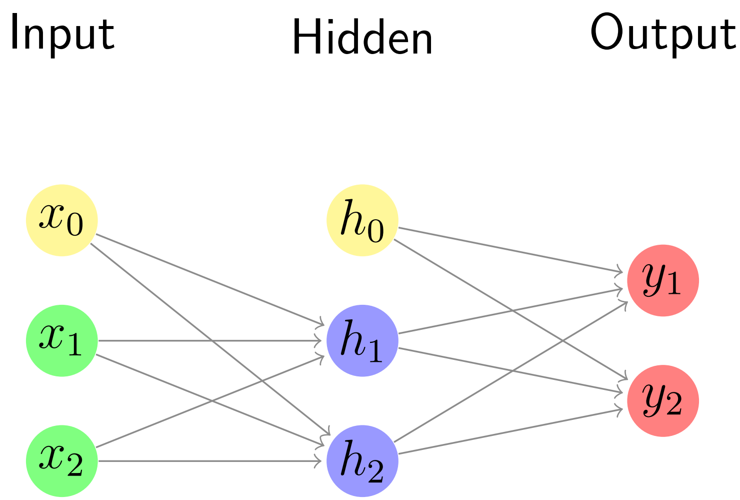
then $\displaystyle \frac{\partial y}{\partial x} = \sum_{i=1}^{n} \frac{\partial y}{\partial u_i} \frac{\partial u_i}{\partial x}$

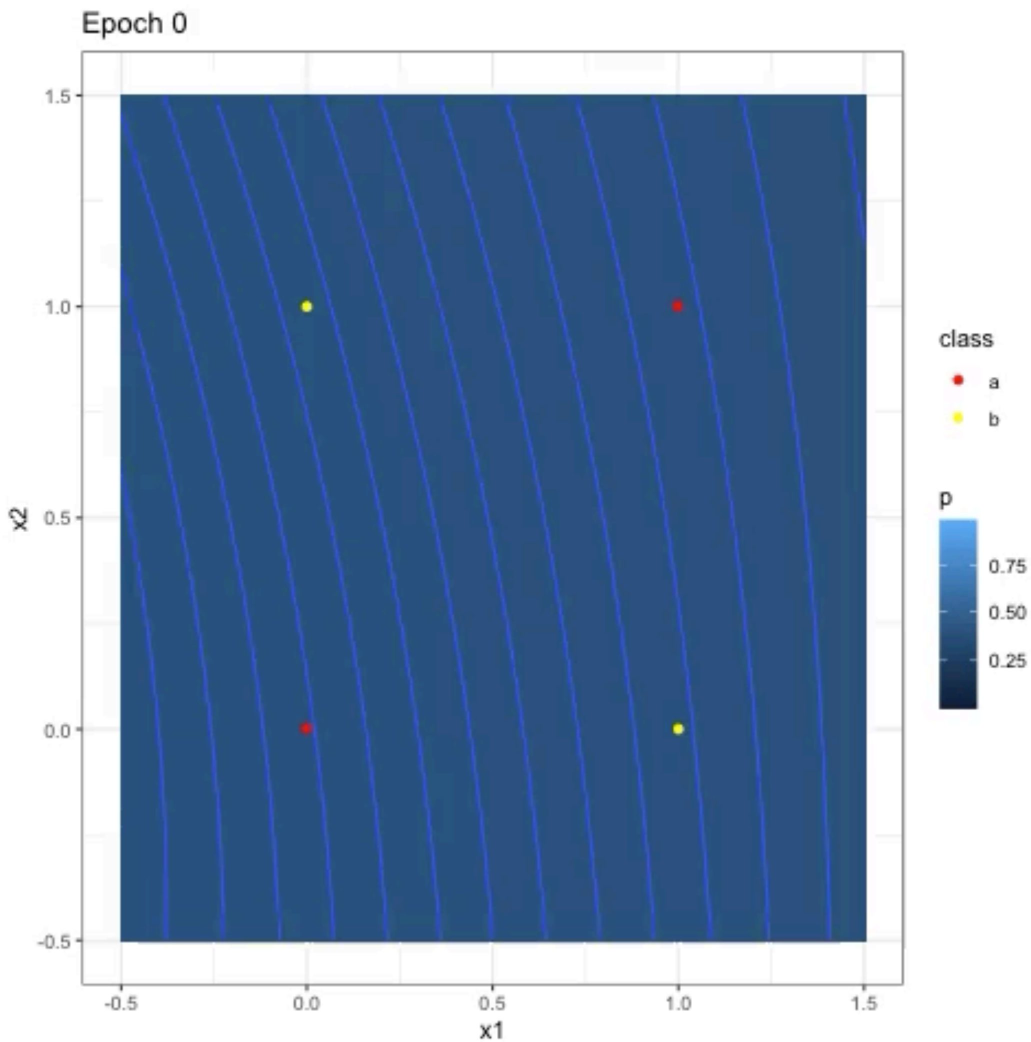
This reuse of partially computed results (here, $\frac{\partial C}{\partial w^\eta_{11}}$ and $\frac{\partial C}{\partial w^\eta_{12}}$) is what is called

BACKPROPAGATION*

(*An instance of **dynamic programming**. Technically, the stored outputs of intermediate computations are not $\frac{\partial C}{\partial w}$ terms themselves, but gradients for node values, from which the weight gradients can be easily computed.)

Learning XOR with one hidden layer

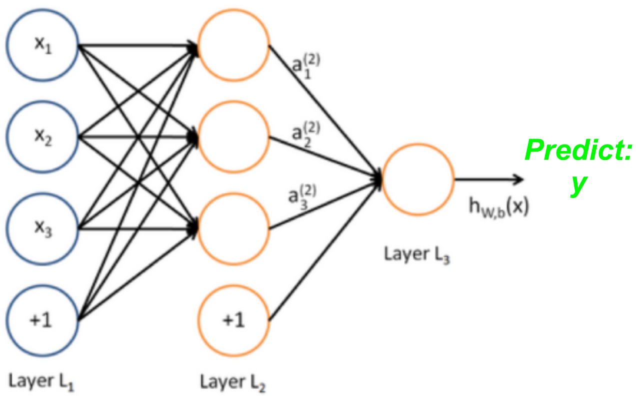




Initialize weights **randomly**

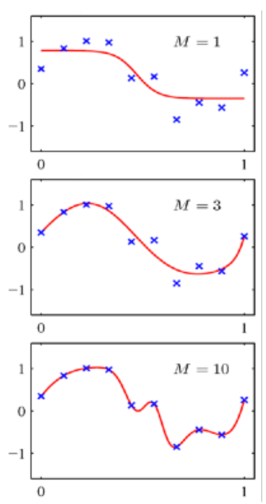
In each learning **epoch**, collect gradient from the 4 datapoints, and move weights "a bit" in direction of gradient

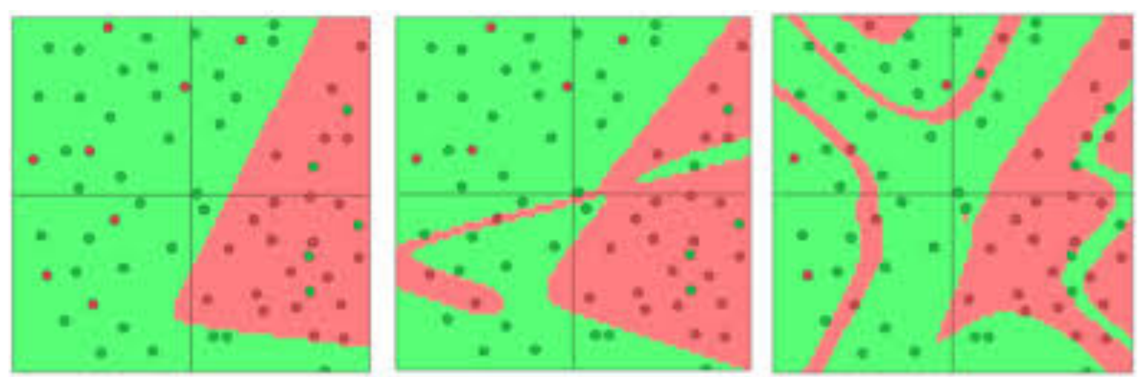
Expressive power of multilayer network



$$g(x_1, \ldots, x_n) = y$$

- Even just one hidden layer makes a neural network a ***universal function approximator*** (Hornik et al., 1989)

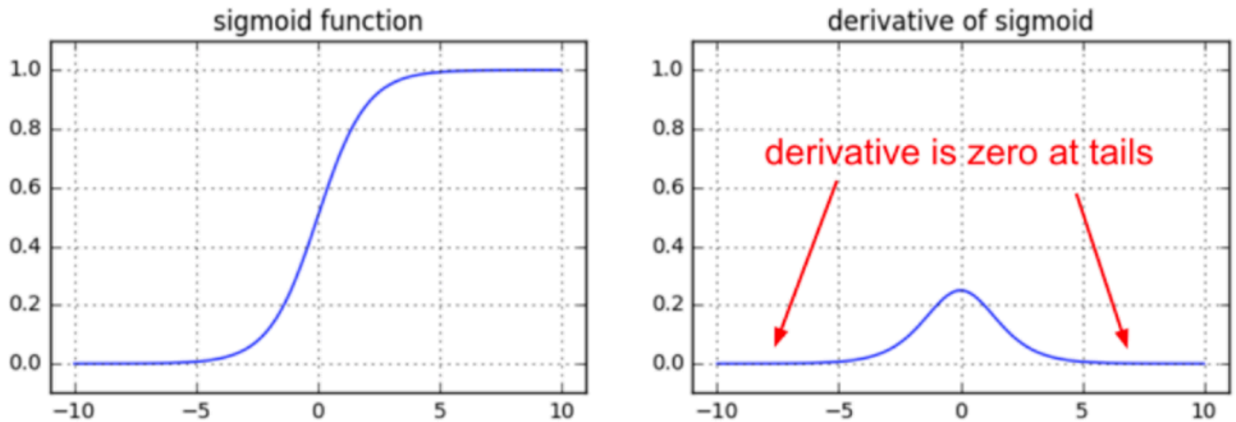




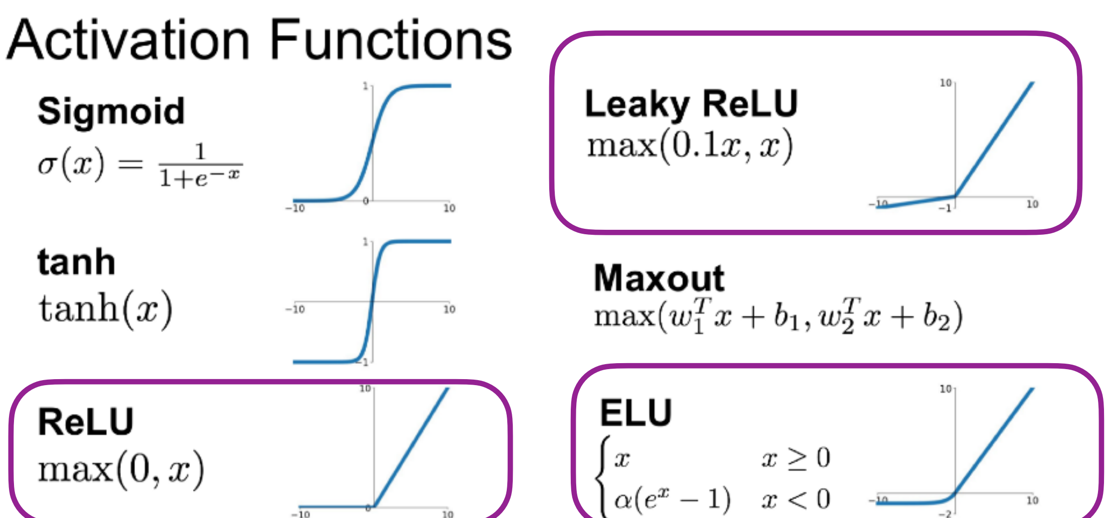
- Challenge: ***how to learn best function approximation?***

Changing activation functions

- Using sigmoid as non-linear activation function $f(\mathbf{H})$ has problems when you add more network layers



- Thus other functions for $f(\mathbf{H})$ have become more popular



Online resources for learning more

Backpropagation:

Backprop as derivatives on computation graphs: http://colah.github.io/posts/2015-08-Backprop/

Lecture by Richard Socher (especially first ~18min) at https://www.youtube.com/watch?v=isPiE-DBagM&list=PL3FW7Lu3i5Jsnh1rnUwq_TcylNr7EkRe6

Worked numerical example: https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

More generally, RNNs in natural language processing:

https://learning-modules.mit.edu/class/index.html?uuid=/course/6/fa17/6.864#info

http://web.stanford.edu/class/cs224n/

http://cs231n.github.io

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Goldberg, Y. (2017). Neural network methods for natural language processing. Synthesis Lectures on Human Language Technologies, 10(1), 1-309. [Available for PDF download through MIT Libraries]

(And if you recommend another resource not listed here, let me know at rplevy@mit.edu!)