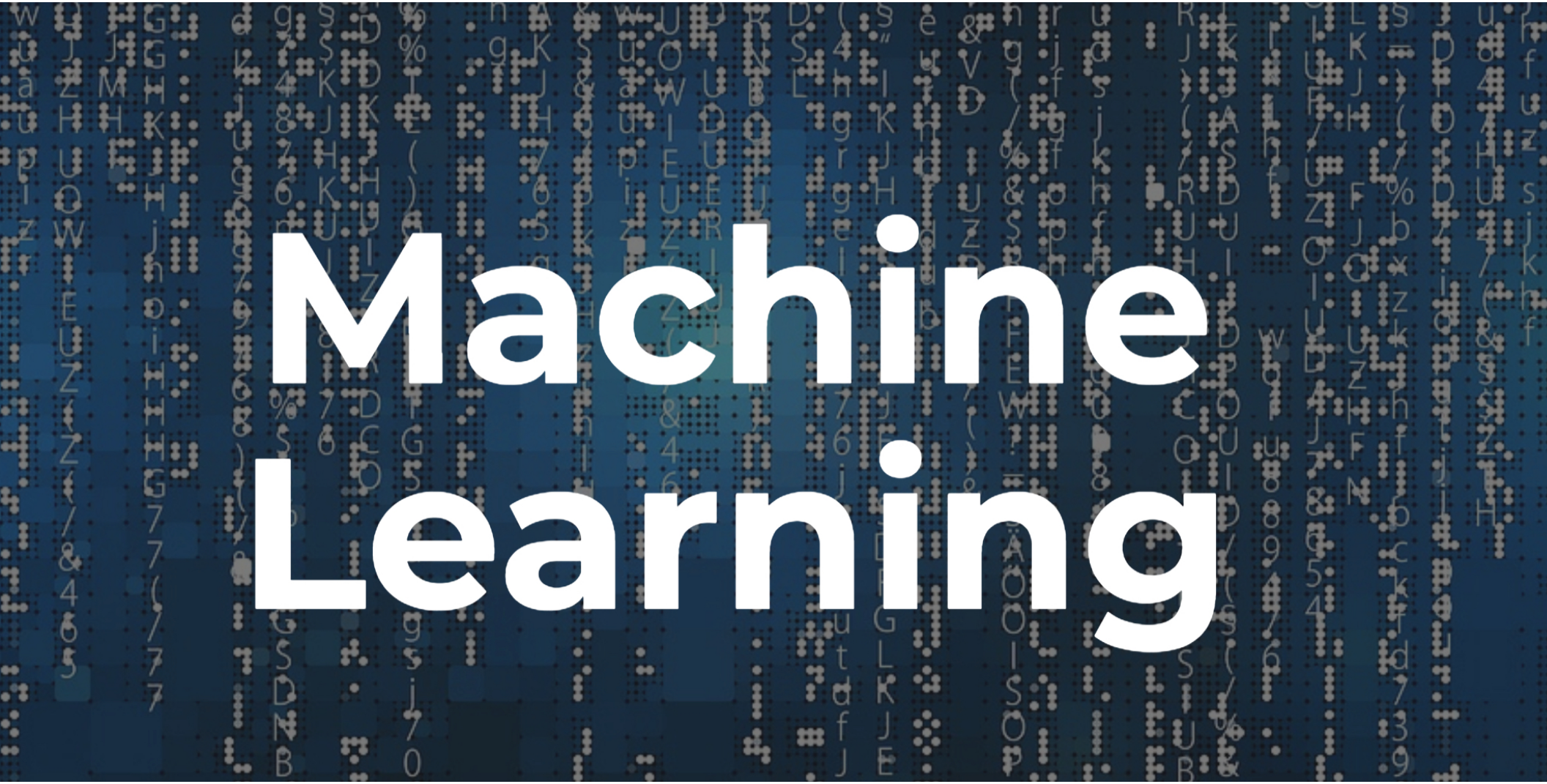


CSCI 5622 Fall 2020

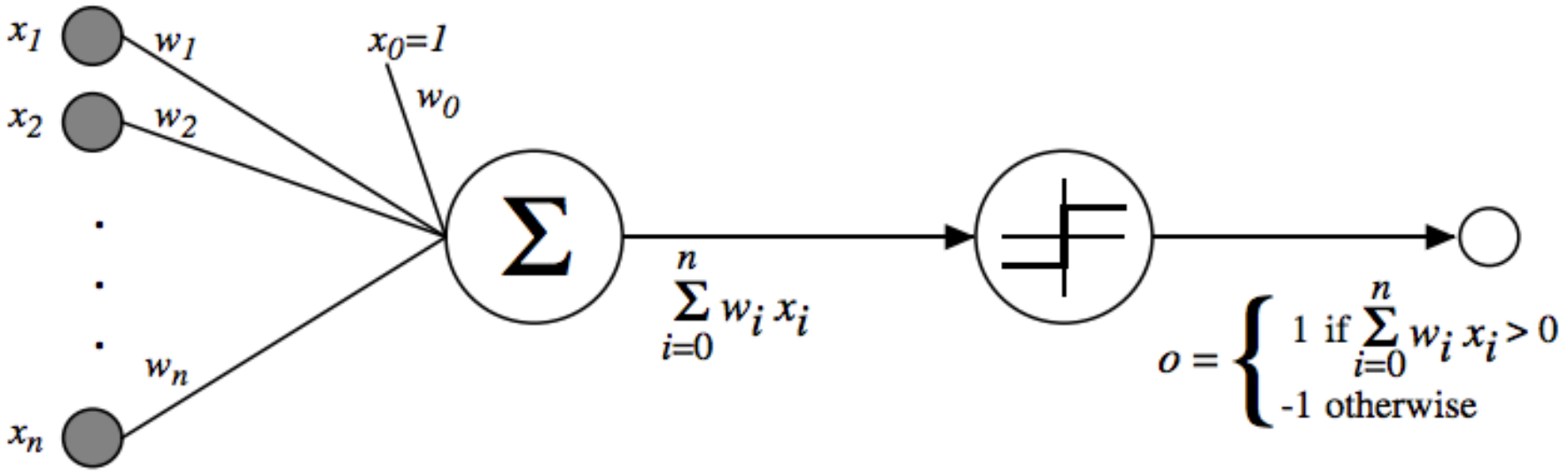
Prof. Claire Monteleoni

Lecture 19:

- Introduction to neural networks and deep learning

With credit to J. Boyd-Graber, C. Ketelson, T. Mitchell, C. Tan

Perceptron

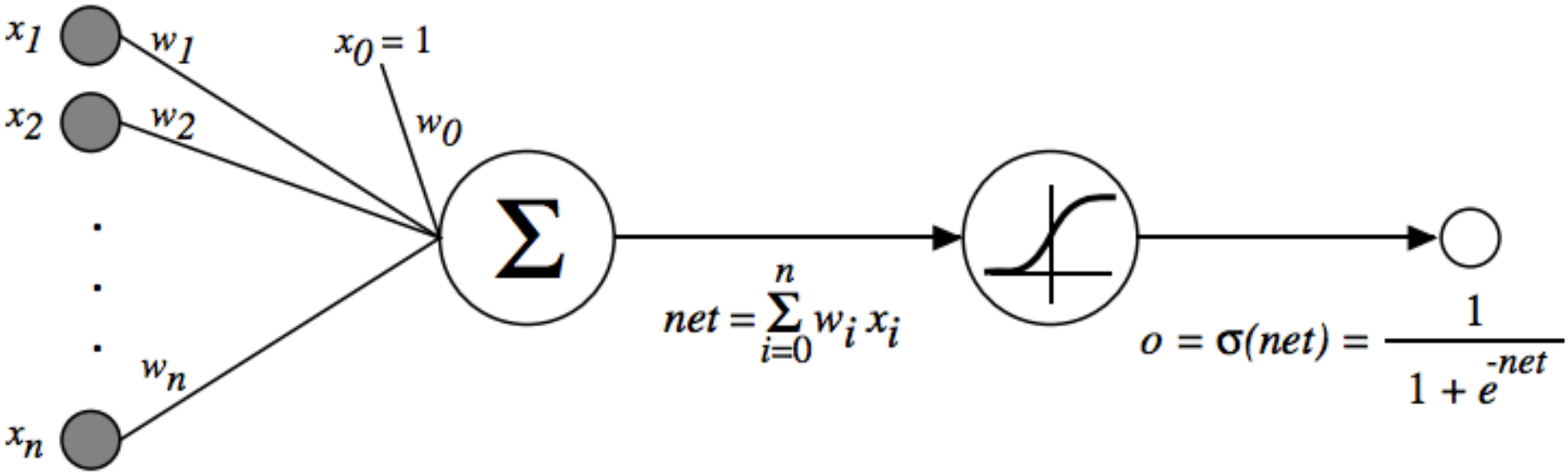


$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Sometimes we'll use simpler vector notation:

$$o(\vec{x}) = \begin{cases} 1 & \text{if } \vec{w} \cdot \vec{x} > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Sigmoid Unit (recall Logistic function)



$\sigma(x)$ is the sigmoid function

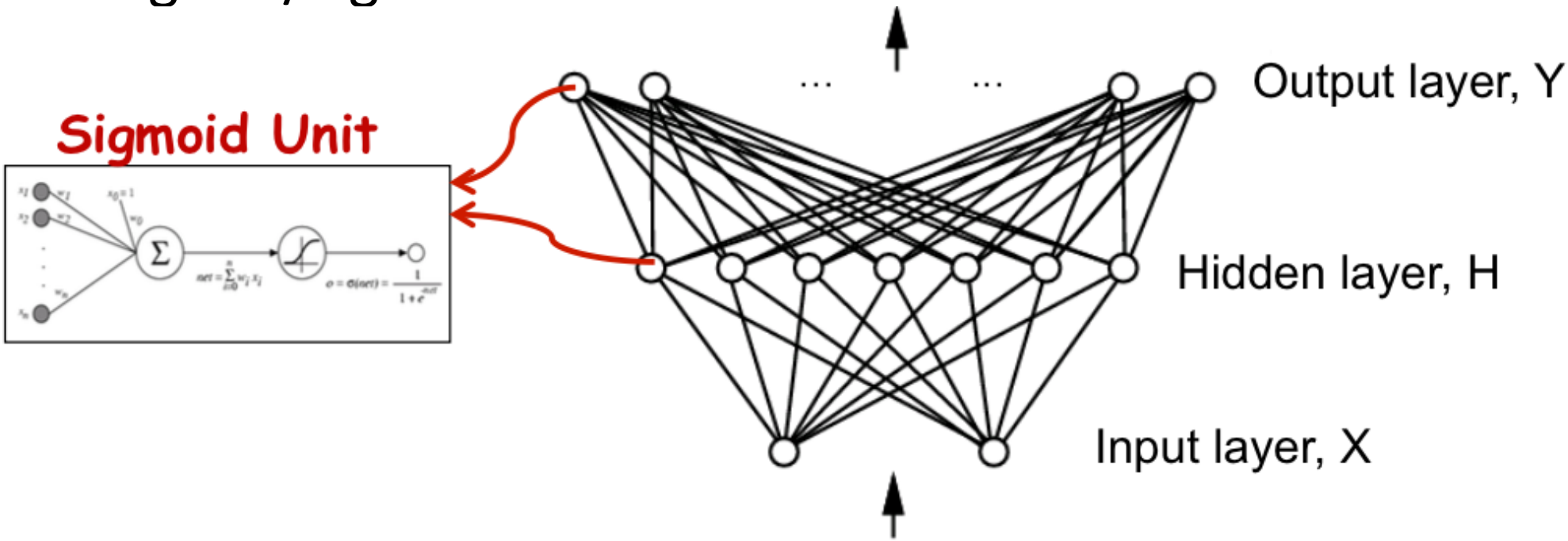
$$\frac{1}{1 + e^{-x}}$$

Neural Network

Learn a function *f: X* → *Y*, where *f* can be non-linear

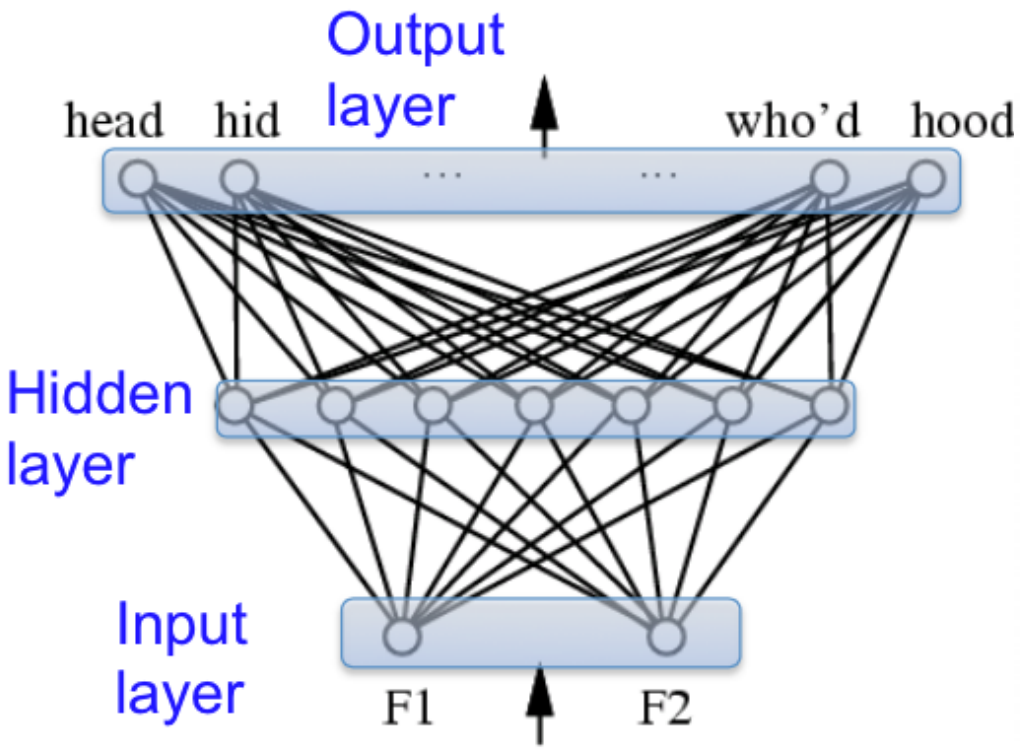
- X (vector of) continuous and/or discrete variables
- Y (vector of) continuous and/or discrete variables

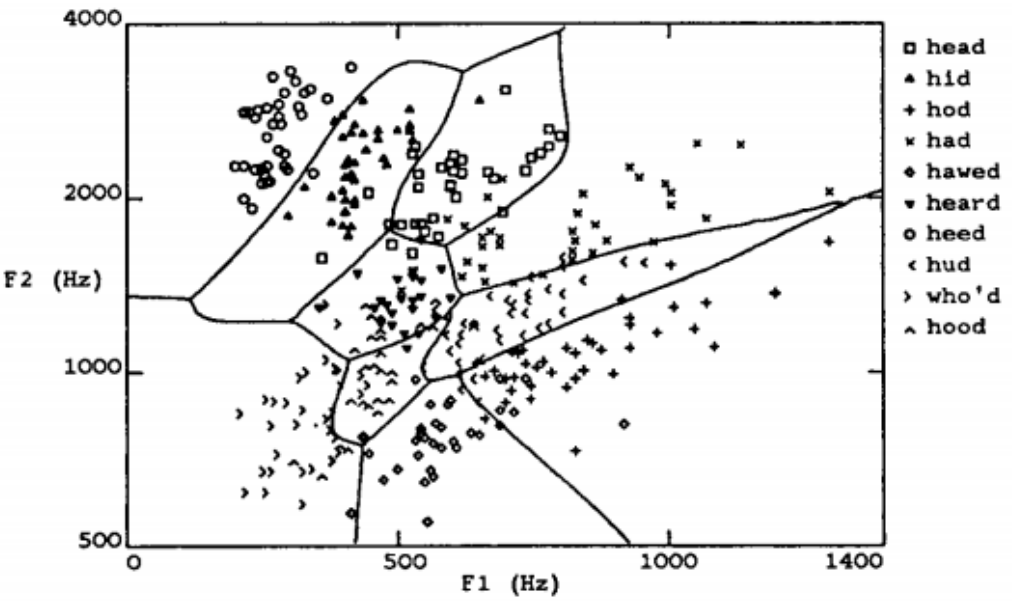
Neural network - Represent *f* by a network of logistic/sigmoid units:



Neural network example

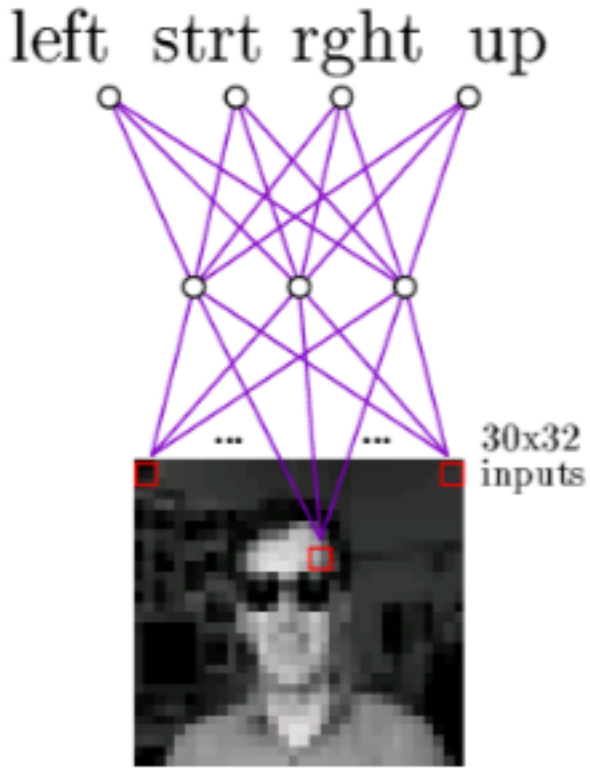
Neural Network trained to distinguish vowel sounds using 2 formants (features)





Two layers of logistic units

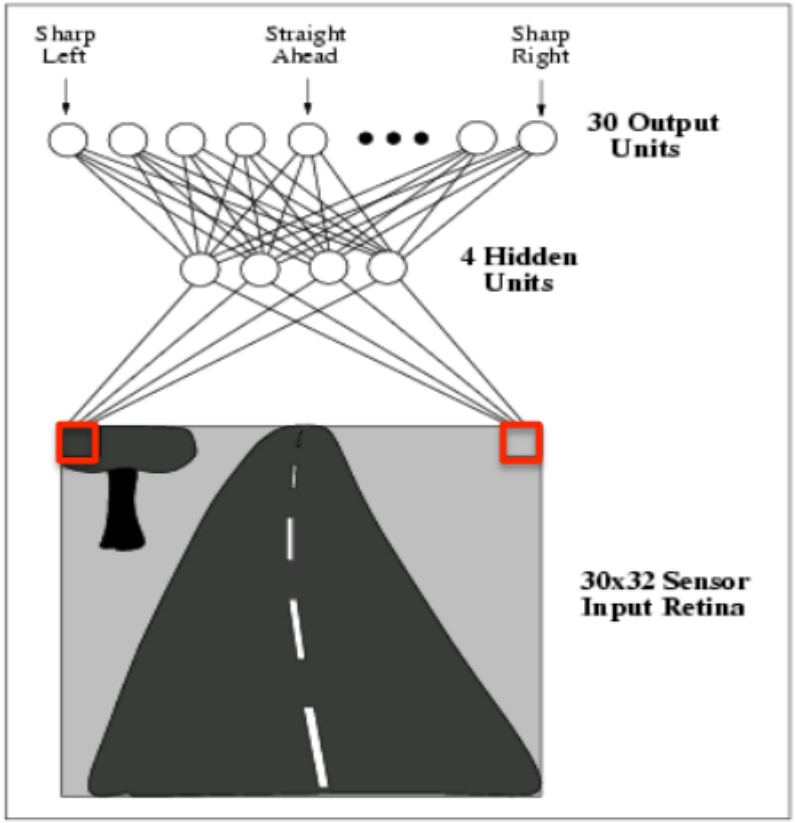
Highly non-linear decision boundary



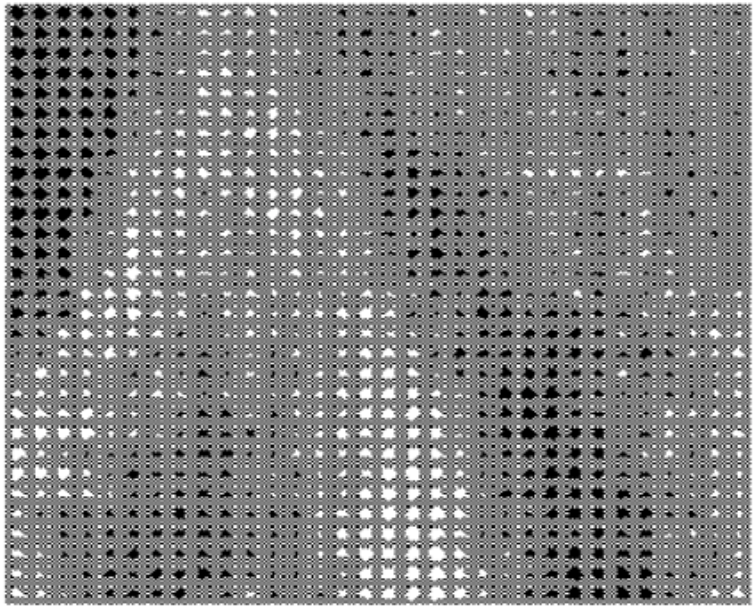
Typical input images

90% accurate learning head pose, and recognizing 1-of-20 faces

Neural Network trained to drive a car!



Weights to output units from the hidden unit



Weights of each pixel for one hidden unit

NN motivations from Human brain

Consider humans:

- Neuron switching time ~ .001 second
- Number of neurons ~ 10^{10}
- Connections per neuron ~ 10^{4-5}
- Scene recognition time ~ .1 second
- 100 inference steps doesn't seem like enough

$\rightarrow$ much parallel computation

Properties of artificial neural nets (ANN's):

- Many neuron-like threshold switching units
- Many weighted interconnections among units
- Highly parallel, distributed process

Prediction with a Neural Network

Prediction: Given neural network (hidden units and weights), use it to predict the label of a test point

Forward Propagation

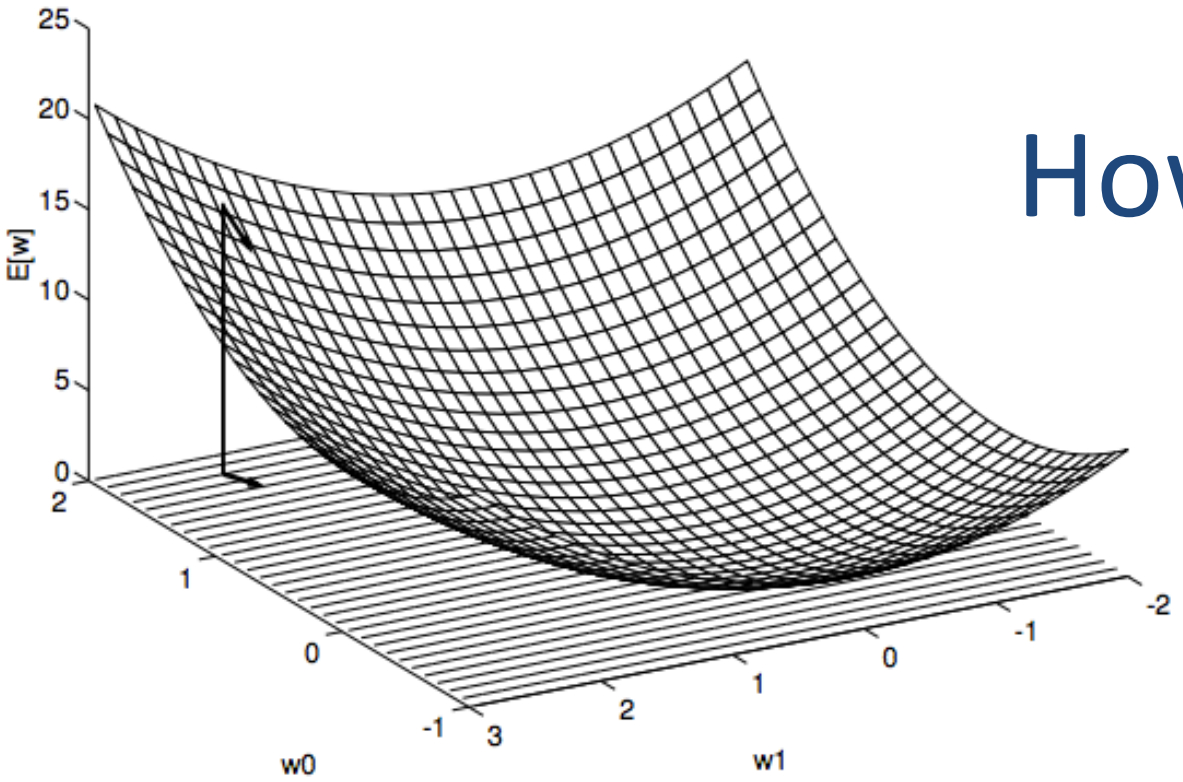
- Start from input layer
- For each subsequent layer, compute output of sigmoid unit

Sigmoid unit:

$$o(\mathbf{x}) = \sigma(w_0 + \sum_i w_i x_i)$$

1-Hidden layer, 1 output NN:

$$o(\mathbf{x}) = \sigma\left(w_0 + \sum_h w_h \underbrace{\sigma(w_0^h + \sum_i w_i^h x_i)}_{o_h}\right)$$



How to train a NN?

Techniques to fit the weights w are motivated by performing gradient descent (with respect to the weights w) on the training error.

The error function may not be convex

Gradient

$$\nabla E[\vec{w}] \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \cdots \frac{\partial E}{\partial w_n}\right]$$

Training rule:

$$\Delta \vec{w} = -\eta \nabla E[\vec{w}]$$

i.e.,

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

Training Neural Networks

$\sigma(x)$ is the sigmoid function

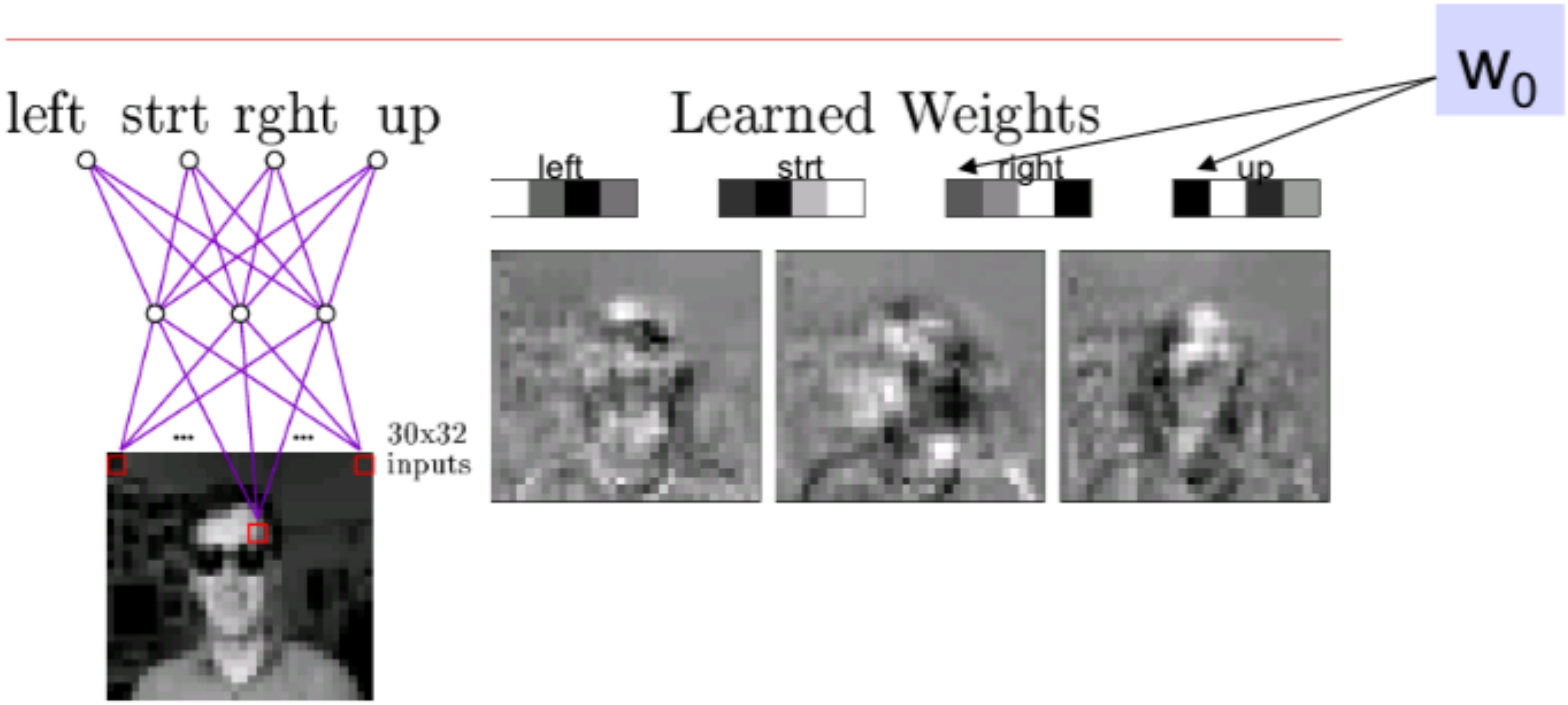
$$\frac{1}{1+e^{-x}}$$

Nice property: $\frac{d\sigma(x)}{dx} = \sigma(x)(1-\sigma(x))$ Differentiable

We can derive gradient decent rules to train

- One sigmoid unit
- *Multilayer networks* of sigmoid units → Backpropagation

Learned Hidden Unit Weights



Typical input images

http://www.cs.cmu.edu/~tom/faces.html

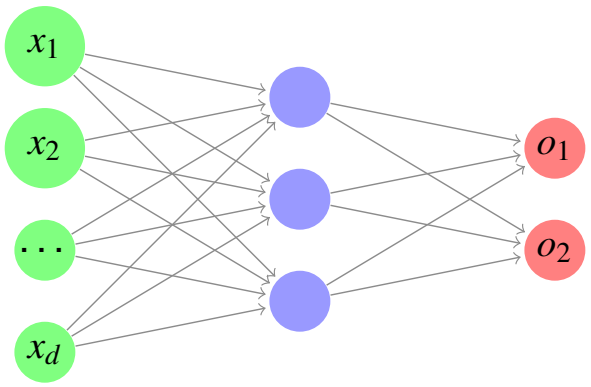
Neural Networks summary

- Actively used to model distributed computation in brain
- Highly non-linear regression/classification
- Vector-valued inputs and outputs
- Potentially millions of parameters to estimate – overfittng
- Hidden layers learn intermediate representations – how many to use?
- Prediction: Forward propagation
- Paramter fitting: Gradient descent (backpropagation), local minima problems
- NN Coming back in new form as deep belief networks

Deep Neural networks

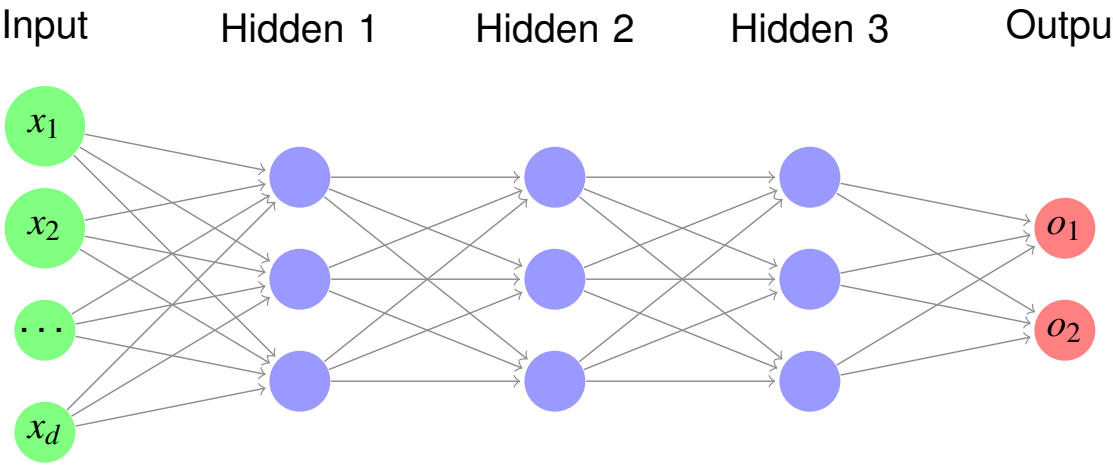
A two-layer example (one hidden layer)

Input Hidden Output



Deep Neural networks

More layers:



Neural networks in a nutshell

- Training data $S_{\text{train}} = \{(\boldsymbol{x}, y)\}$
- Network architecture (model)

$$\hat{y} = f_w(\boldsymbol{x})$$

- Loss function (objective function)

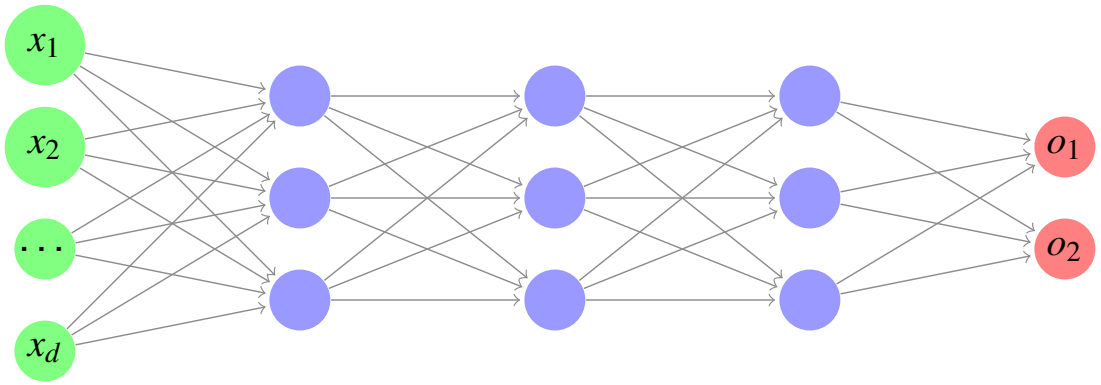
$$\mathscr{L}(y, \hat{y})$$

- Learning (we'll get to this later)

Forward propagation algorithm

How do we make predictions based on a multi-layer neural network?
Store the biases for layer l in $\boldsymbol{b}^l$, weight matrix in $\boldsymbol{W}^l$

$\boldsymbol{W}^1, \boldsymbol{b}^1$ $\boldsymbol{W}^2, \boldsymbol{b}^2$ $\boldsymbol{W}^3, \boldsymbol{b}^3$ $\boldsymbol{W}^4, \boldsymbol{b}^4$



Forward propagation algorithm

Suppose your network has L layers.
How to predict the label of a test data point, $\boldsymbol{x}$:

1: Initialize $\boldsymbol{a}^0 = \boldsymbol{x}$
2: **for** $l = 1$ to L **do**
3: $\quad \boldsymbol{z}^l = \boldsymbol{W}^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l$
4: $\quad \boldsymbol{a}^l = g(\boldsymbol{z}^l)$
5: **end for**
6: The prediction $\hat{y}$ is simply $\boldsymbol{a}^L$

Nonlinearity

What happens if there is no nonlinearity?

Nonlinearity

What happens if there is no nonlinearity?
Linear combinations of linear combinations are still linear combinations.

Nonlinearity Options

- Sigmoid

$$f(x) = \frac{1}{1 + \exp(x)}$$

- tanh

$$f(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$

- ReLU (rectified linear unit)

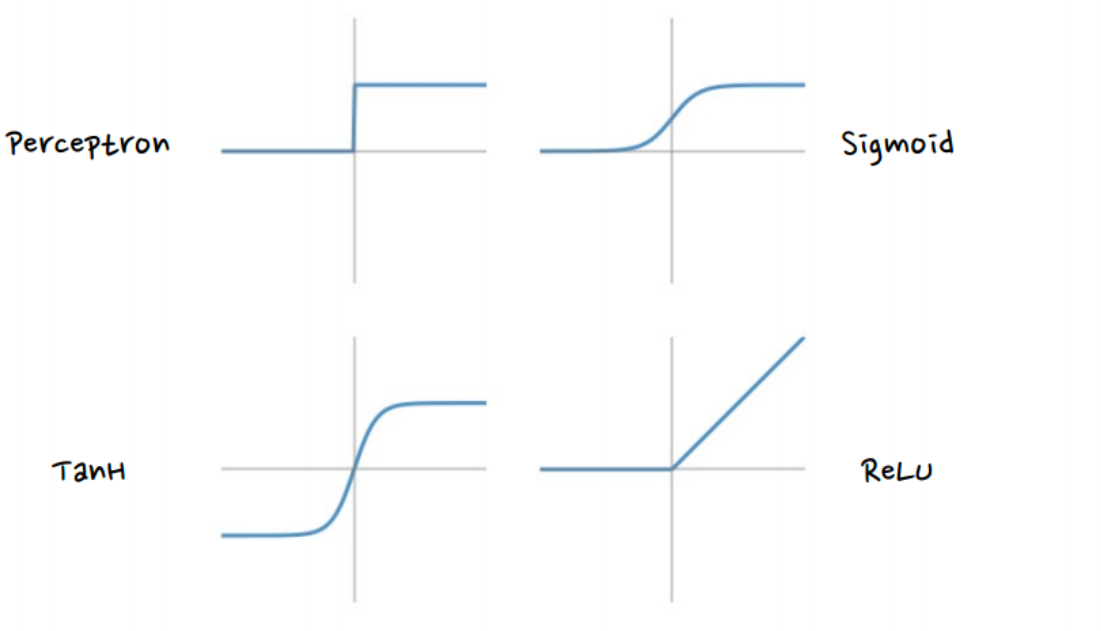
$$f(x) = \max(0, x)$$

- softmax

$$\boldsymbol{x} = \frac{\exp(\boldsymbol{x})}{\sum_{x_i} \exp(x_i)}$$

`https://keras.io/activations/`

Nonlinearity Options



Loss Function Options

- ℓ_2 loss

$$\sum_i (y_i - \hat{y}_i)^2$$

- ℓ_1 loss

$$\sum_i |y_i - \hat{y}_i|$$

- Cross entropy

$$-\sum_i y_i \log \hat{y}_i$$

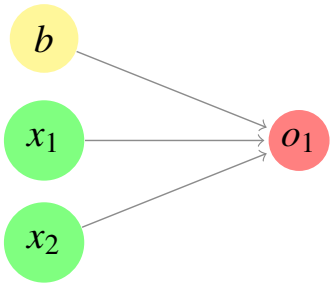
- Hinge loss

$$\max(0, 1 - y\hat{y})$$

`https://keras.io/losses/`

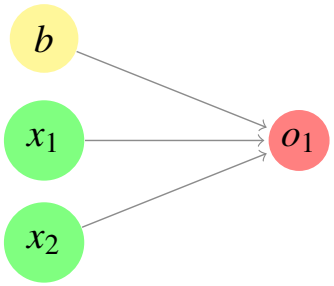
A Perceptron Example

$\boldsymbol{x} = (x_1, x_2), y = f(x_1, x_2)$



A Perceptron Example

$\boldsymbol{x} = (x_1, x_2), y = f(x_1, x_2)$



We consider a simple activation function

$$f(z) = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases}$$

A Perceptron Example

Simple Example: Can we learn OR?

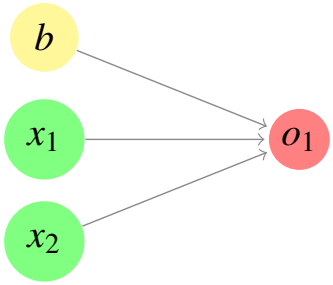
x_1	0	1	0	1
x_2	0	0	1	1
$y = x_1 \vee x_2$	0	1	1	1

A Perceptron Example

Simple Example: Can we learn OR?

x_1	0	1	0	1
x_2	0	0	1	1
$y = x_1 \vee x_2$	0	1	1	1

$$\boldsymbol{w} = (1, 1), b = -0.5$$



A Perceptron Example

Simple Example: Can we learn AND?

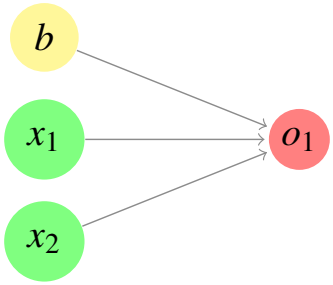
x_1	0	1	0	1
x_2	0	0	1	1
$y = x_1 \wedge x_2$	0	0	0	1

A Perceptron Example

Simple Example: Can we learn AND?

x_1	0	1	0	1
x_2	0	0	1	1
$y = x_1 \wedge x_2$	0	0	0	1

$$\boldsymbol{w} = (1, 1), b = -1.5$$



A Perceptron Example

Simple Example: Can we learn NAND?

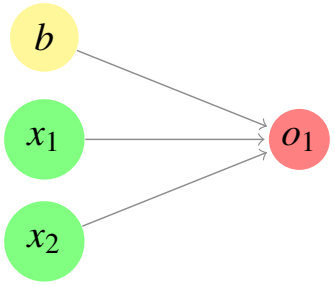
x_1	0	1	0	1
x_2	0	0	1	1
$y = \neg(x_1 \wedge x_2)$	1	0	0	0

A Perceptron Example

Simple Example: Can we learn NAND?

x_1	0	1	0	1
x_2	0	0	1	1
$y = \neg(x_1 \wedge x_2)$	1	0	0	0

$$\boldsymbol{w} = (-1, -1), b = 0.5$$



A Perceptron Example

Simple Example: Can we learn XOR?

x_1			0	1	0	1
x_2			0	0	1	1
x_1	XOR	x_2	0	1	1	0

A Perceptron Example

Simple Example: Can we learn XOR?

x_1			0	1	0	1
x_2			0	0	1	1
x_1	XOR	x_2	0	1	1	0

NOPE!

A Perceptron Example

Simple Example: Can we learn XOR?

x_1			0	1	0	1
x_2			0	0	1	1
x_1	XOR	x_2	0	1	1	0

NOPE!
But why?

A Perceptron Example

Simple Example: Can we learn XOR?

x_1			0	1	0	1
x_2			0	0	1	1
x_1	XOR	x_2	0	1	1	0

NOPE!
But why?
The single-layer perceptron is just a linear classifier, and can only learn things that are linearly separable.

A Perceptron Example

Simple Example: Can we learn XOR?

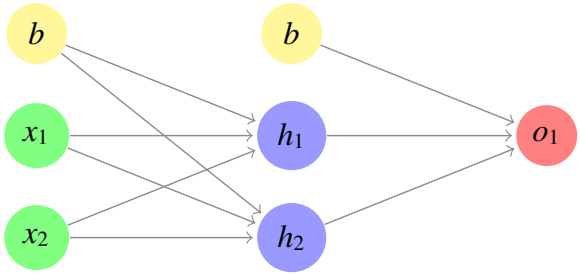
x_1			0	1	0	1
x_2			0	0	1	1
x_1	XOR	x_2	0	1	1	0

NOPE!
But why?
The single-layer perceptron is just a linear classifier, and can only learn things that are linearly separable.
How can we fix this?

A Perceptron Example

Increase the number of layers.
(Solution due to R. Grosse uses inputs in $\{-1, 1\}$)

x_1	-1	1	-1	1
x_2	-1	-1	1	1
x_1 XOR x_2	-1	1	1	-1



$$\boldsymbol{W}^1 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \boldsymbol{b}^1 = \begin{bmatrix} -0.5 \\ -1.5 \end{bmatrix}$$

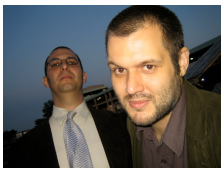
$$\boldsymbol{W}^2 = \begin{bmatrix} 1 & -1 \end{bmatrix}, \boldsymbol{b}^2 = -0.5$$

General Expressiveness of Neural Networks

Neural networks with a single hidden layer can approximate any measurable functions [Hornik et al. 1989, Cybenko 1989].

Outline

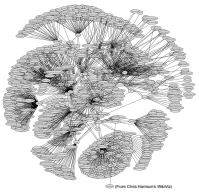
Feature Engineering



$\longrightarrow \langle 1.5, 3.2, -5.1, \ldots, 4.2 \rangle$

Republican nominee George Bush said he felt nervous as he voted today in his adopted home state of Texas, where he ended...

$\longrightarrow \langle 1, 0, 0, 0, 5, 0, 9, 3, 1, \ldots, 0 \rangle$



$\longrightarrow \begin{bmatrix} 1 & 0 & 1 & \ldots & 0 \\ 0 & 1 & 1 & \ldots & 0 \\ 1 & 0 & 0 & \ldots & 1 \\ & \ldots & & & \\ 0 & 0 & 0 & \ldots & 0 \end{bmatrix}$

Outline

Revisiting Logistic Regression

$$P(\mathrm{Y}=0 \mid \boldsymbol{x}, \beta) = \frac{1}{1+\exp\left[\beta_0 + \sum_i \beta_i X_i\right]}$$

$$P(\mathrm{Y}=1 \mid \boldsymbol{x}, \beta) = \frac{\exp\left[\beta_0 + \sum_i \beta_i X_i\right]}{1+\exp\left[\beta_0 + \sum_i \beta_i X_i\right]}$$

$$\mathscr{L} = -\sum_j \log P(y^{(j)} \mid X^{(j)}, \beta)$$

Revisiting Logistic Regression

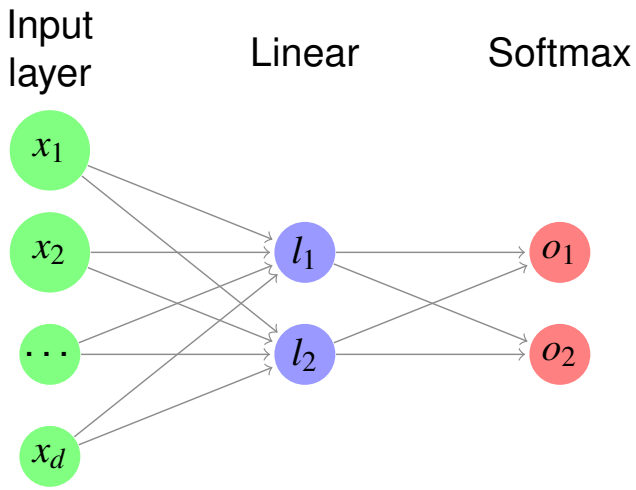
- Transformation on $\boldsymbol{x}$ (we map class labels from $\{0, 1\}$ to $\{1, 2\}$):

$$l_i = \beta_i^T \boldsymbol{x}, i = 1, 2$$
$$o_i = \frac{\exp l_i}{\sum_{c \in \{1,2\}} \exp l_c}, i = 1, 2$$

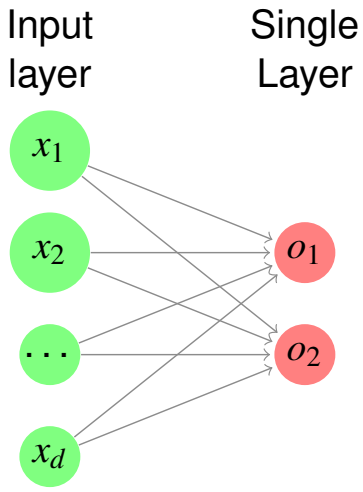
- Objective function (using cross entropy $-\sum_i p_i \log q_i$):

$$\mathscr{L}(Y, \hat{Y}) = -\sum_j P(y^{(j)} = 1) \log P(\hat{y}_i = 1 \mid x^{(j)}, \beta) + P(y^{(j)} = 0) \log \hat{P}(y_i = 0 \mid X_i)$$

Logistic Regression as a Single-layer Neural Network



Logistic Regression as a Single-layer Neural Network



Outline

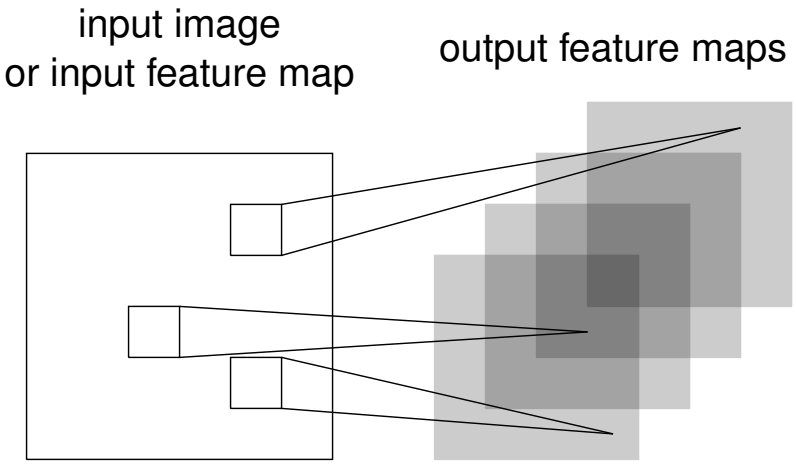
Structured data

Spatial information

https://www.reddit.com/r/aww/comments/6ip2la/before_and_after_she_was_told_she_was_a_good_girl/

Convolutional Layers

Sharing parameters across patches



https://github.com/davidstutz/latex-resources/blob/master/tikz-convolutional-layer/convolutional-layer.tex

Structured data

Sequential information, e.g.,

- language
- activity history
- weather measurements
- stock market data streams

$$\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T)$$

Recurrent Layers

Sharing parameters along a sequence

$$h_t = f(x_t, h_{t-1})$$

Recurrent Layers

Sharing parameters along a sequence

$$h_t = f(x_t, h_{t-1})$$

Long short-term memory

What is missing?

- How to find good weights?
- How to make the model work (regularization, architecture, etc)?