University of California
San Francisco

BMI-203: Biocomputing Algorithms
Lecture 7: Machine Learning I

Michael J Keiser, PhD
Assistant Professor
Pharmaceutical Chemistry
Bioengineering and Therapeutic Sciences
Institute for Neurodegenerative Diseases

keiser@keiserlab.org

Outline

- Today: Machine learning fundamentals
  - Supervised learning
  - Linear Regression
  - Logistic Regression
  - Perceptron learning algorithm
  - Multilayer neural networks

- Next week: Nearest neighbors, evaluating performance, unsupervised learning, deep learning

Machine learning resources

- Theory and background
  - http://ufldl.stanford.edu/tutorial/
  - http://cs229.stanford.edu/materials.html
  - http://web.stanford.edu/class/cs221/
  - http://openclassroom.stanford.edu/MainFolder/CoursePage.php?course=MachineLearning

- Rapid implementation libraries and tools
  - http://deeplearning.net/software/theano/tutorial/
  - http://deeplearning4j.org/quickstart.html

Supervised vs. unsupervised learning

- Supervised (today)
  - "Correct" answers given (data).
  - Regression: Continuous value prediction (housing prices).
  - Classification: Discrete value prediction (tumor malignancy).

- Unsupervised
  - Part of the task is to define the categories in the first place.
  - Clustering: gene microarrays, social networks, market segmentation.
  - E.g., Cocktail party algorithm.

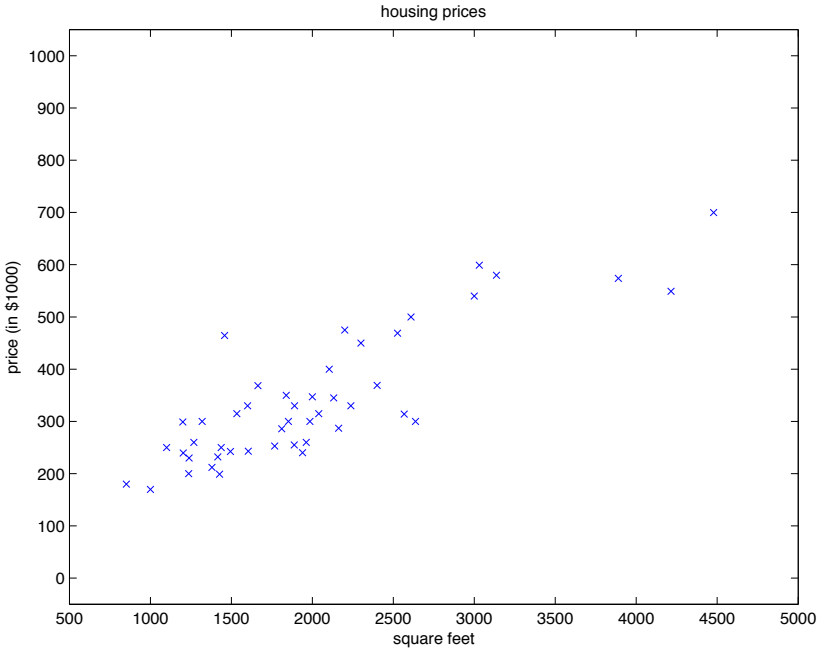
Four things to do for a machine-learning task

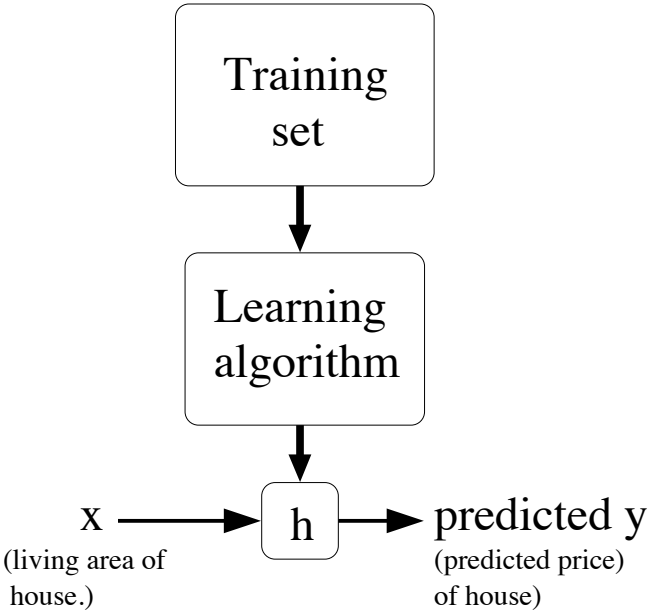
1. Choose a representation of your input data
2. Choose a functional form that will map input examples to outputs
3. Choose a method of optimization
4. Train your system and evaluate performance
   - Ideal method: blind testing of a trained classifier on new data
   - Other method: cross-validation
     - Train your system on all but a subset of your data
     - Test on the held out subset
     - Repeat to get an estimate of predictive performance

Source: Ajay Jain

Supervised learning

- Example: Given data like these, how can we learn to predict prices of houses as a function of living areas?





Source: CS229 Notes, Andrew Ng (Stanford)

Four things to do for a machine-learning task: linear regression

1. Choose a representation of your input data
   - ✓ (House size in square feet x, house price in thousands of dollars y)
2. Choose a functional form that will map input examples to outputs
   - ✓ Linear combination of weights θ_i
3. Choose a method of optimization
   - ✓ Gradient descent
4. Train your system and evaluate performance
   - Ideal method: blind testing of a trained classifier on new data
   - Other method: cross-validation
   - ✓ In this simple example we'll train on all data and won't evaluate performance via either method.
     - But beware this puts us in danger of over-fitting!!

Source: Ajay Jain

Review: Linear regression

- Goal: Find $y = h(x)$ so that we have $y^{(i)} \approx h(x^{(i)})$ for each example *i*

- Let's use a linear function: $h_\theta = \theta_0(x_0) + \theta_1 x_1 + \theta_2 x_2 + \ldots$
  - Then: $h_\theta(x) = \sum_j \theta_j x_j$
  - This is called a "hypothesis class"

- We want to find a choice of weights $\theta = [\theta_0, \theta_1, \theta_2, \ldots]$ that minimizes the cost function:

$$J(\theta) = \frac{1}{2} \sum_i \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

x_i	input variables (features)
y_i	output (target) variables
$(x^{(i)},y^{(i)})$	i^{th} training example
θ_i	parameters (aka weights)
x_0	the intercept: $x_0 = 1$

Source: UFLDL Tutorial, Andrew Ng (Stanford)

Gradient descent for linear regression

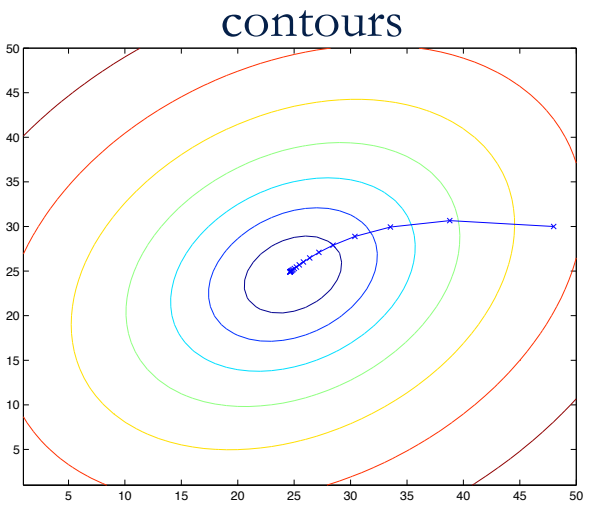
- Start with some values θ and take downhill steps in their parameter space.
- Per the optimization lecture, we can use gradient descent to minimize cost function,

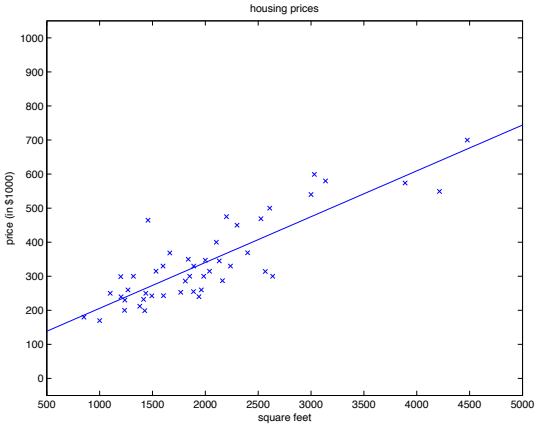
$$J(\theta) = \frac{1}{2} \sum_i \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

- However we need to calculate its gradient first:

$$\nabla_\theta J(\theta) = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \frac{\partial J(\theta)}{\partial \theta_2} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{bmatrix} \Rightarrow \frac{\partial J(\theta)}{\partial \theta_j} = \sum_i x_j^{(i)} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

j	denotes input variable
$(x^{(i)}, y^{(i)})$	i^{th} training example





Source: UFLDL Tutorial & CS229 Notes, Andrew Ng (Stanford)

UCSF

Regression as an optimization algorithm

- Goal: Minimize $J(\theta)$ over choices of θ

1. Initialize weights $\theta = [\theta_0, \theta_1, \theta_2] = \textit{vector of small random values}$
2. Define a cost function, $J(\theta)$
3. Define a gradient, $\nabla_\theta J(\theta)$
4. For $t = 1, \ldots, T$:
   - Set $\theta = \theta - \alpha \nabla_\theta J(\theta)$
   - Halt on convergence and report θ

What does α do?

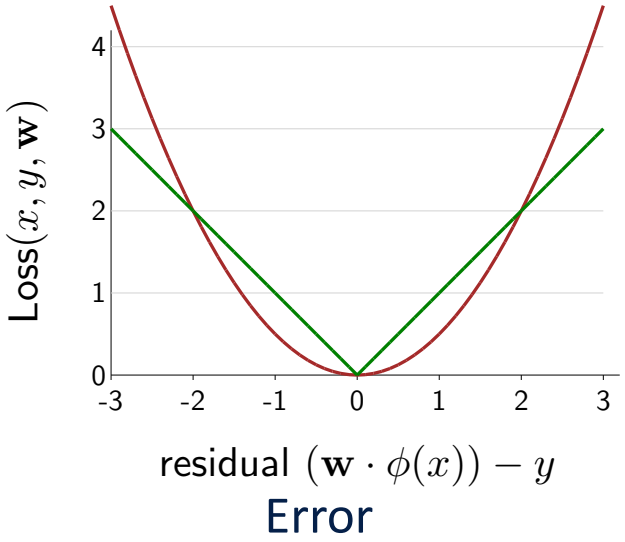
- Stochastic gradient descent: Use *one random* training example per time step.
- You could also use another optimization approach, like Newton's Method, but we'll focus on gradient descent here, as it's relevant to back-propagation (later).

Cost functions (aka loss functions)

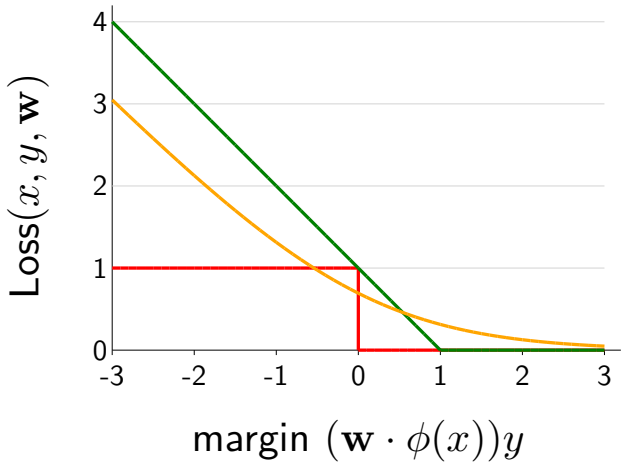
- So far, we've focused on mean squared error, for linear regression.
- But it's not the only game in town.

(We'll address classification a bit later)

Regression



Binary classification



Syntax note: $\mathbf{w} = \theta$, and $\phi(x)$ is an optional transform of x

More info: CS221 AI Principles & Techniques, Percy Liang (Stanford)

UCSF

In practice – rates & training time

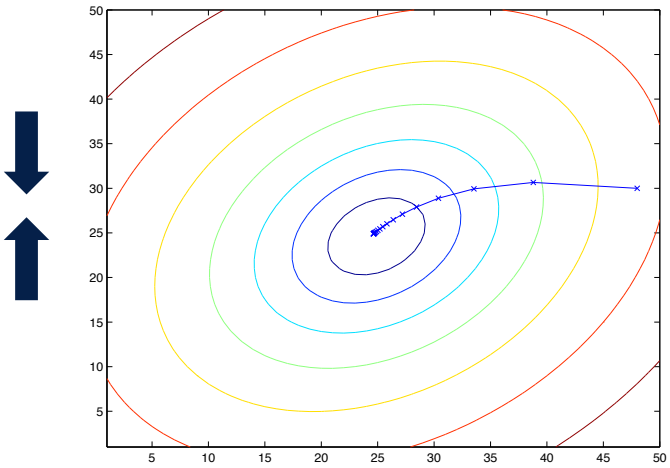
- Learning rate
  - Gradient descent's learning rate (α) is a tradeoff between reasonable speed and ability to converge.
    - For any given α, plot iterations vs. $J(\theta)$
      - For linear regression with least squares, it should *always* decrease.
    - Why? And what's going on if it doesn't?
    - Response: Sample a range of α, increasing ~3x per step.

- For speed, all this is typically all done in vectorized and matrix formats.
  - E.g., $\theta^T x = \theta_0 + \sum_{j=1}^{n} \theta_j x_j$

More info: OpenClassroom lectures, Andrew Ng (Stanford)

In practice - features

- Feature scaling
  - Gradient descent only really works when features x_i are on similar scales.
    - Why?
    - Response: Normalize all x_1 to $(0\text{-}1)$, or convert to z-scores.



More info: OpenClassroom lectures, Andrew Ng (Stanford)

What about interactions among features?

- Example: health prediction
  - Input: Patient info x *(a vector)*
  - Output: A health score y

$$\phi(x) = [height(x), weight(x)]$$

- Problem: Can't capture relationship between height and weight

More info: CS221 AI Principles & Techniques, Percy Liang (Stanford)

Interaction between features: Attempt 2

- Solution: Define features that nonlinearly combine inputs
  - E.g., you could take squared difference between expected (from medical formula) vs observed weight:

$$\phi(x) = (52 + 1.9(height(x)\text{-}60) - weight(x))^2$$

- Problem: Requires manually-specified domain knowledge

More info: CS221 AI Principles & Techniques, Percy Liang (Stanford)

Interaction between features: Attempt 3

- Solution: Add lots of features combining multiple measurements

$$\phi(x) = [1, height(x), weight(x), height(x)^2, weight(x)^2, height(x)weight(x)]$$

- Breaks single complex feature (previous attempt) into simpler building blocks

Linear in what?

- Can we obtain decision boundaries of the input that are circles?
- Recall, the prediction's driven by the score:

$$h_\theta = \theta\, \phi(x)$$
$$h_\theta = \theta_0\_(x_0) + \theta_1\phi(x_1) + \theta_2\phi(x_2), x_0 = 1$$

  - Linear in θ? …Yes
  - Linear in $\phi(x)$? …Yes
  - Linear in x? …No (x doesn't even have to be a vector)
- Key idea: Our predictor can be expressive of non-linear functions and decision boundaries of x
- Practical caution: Features can take on vastly different value ranges
  - Rescaling is essential

More info: CS221 AI Principles & Techniques, Percy Liang (Stanford)

Representation of input data can have a big impact on your function and on inductive bias

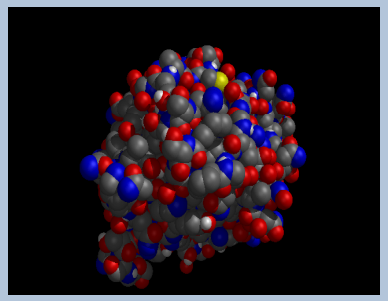
- Does the representation encode the object completely?
  - Enough for the function you care about?
  - Can it accommodate transformations and noise?
- Does the representation encode the object compactly?
  - Is there a well-defined measure of distance between representations that is correlated with outcome?
- If so, you're probably in good shape
- How dependent are we on domain knowledge?

```
ACCACCATGA ATCCACTCCT GATCCTTACC TTTGTGGCAG CTGCTCTTGC TGCCCCCTTT
GATGATGATG ACAAGATCGT TGGGGGCTAC AACTGTGAGG AGAATTCTGT CCCCTACCAG
GTGTCCCTGA ATTCTGGCTA CCACTTCTGT GGTGGCTCCC TCATCAACGA ACAGTGGGTG
GTATCAGCAG GCCACTGCTA CAAGTCCCGC ATCCAGGTGA GACTGGGAGA GCACAACATC
GAAGTCCTGG AGGGGAATGA GCAGTTCATC AATGCAGCCA AGATCATCCG CCACCCCCAA
TACGACAGGA AGACTCTGAA CAATGACATC ATGTTAATCA AGCTCTCCTC ACGTGCAGTA
ATCAACGCCC GCGTGTCCAC CATCTCTCTG CCCACCGCCC CTCCAGCCAC TGGCACGAAG
TGCCTCATCT CTGGCTGGGG CAACACTGCG AGCTCTGGCG CCGACTACCC AGACGAGCTG
CAGTGCCTGG ATGCTCCTGT GCTGAGCCAG GCTAAGTGTG AAGCCTCCTA CCCTGGAAAG
ATTACCAGCA ACATGTTCTG TGTGGGCTTC CTTGAGGGAG GCAAGGATTC ATGTCAGGGT
GATTCTGGTG GCCCTGTGGT CTGCAATGGA CAGCTCCAAG GAGTTGTCTC CTGGGGTGAT
GGCTGTGCCC AGAAGAACAA GCCTGGAGTC TACACCAAGG TCTACAACTA CGTGAAATGG
ATTAAGAACA CCATAGCTGC CAATAGCTAA AGCCCCCAGT ATCTCTTCAG TCTCTATACC
AATAAAGTGA CCCTGTTCTC
```

DNA sequence of Human Trypsin

```
MNPLLILTFV AAALAAPFDD DDKIVGGYNC EENSVPYQVS LNSGYHFCGG SLINEQWVVS
AGHCYKSRIQ VRLGEHNIEV LEGNEQFINA AKIIRHPQYD RKTLNNDIML IKLSSRAVIN
ARVSTISLPT APPATGTKCL ISGWGNTASS GADYPDELQC LDAPVLSQAK CEASYPGKIT
SNMFCVGFLE GGKDSCQGDS GGPVVCNGQL QGVVSWGDGC AQKNKPGVYT KVYNYVKWIK
NTIAANS
```

AA sequence of Human Trypsin



3D structure of Human Trypsin

Source: Ajay Jain

Four things to do for a machine-learning task: logistic regression

1. Choose a representation of your input data
   - ✓ (tumor size x, tumor malignancy $y \; in \; \{0,1\}$)
2. Choose a functional form that will map input examples to outputs
   - ✓ Linear combination of weights θ_i "squashed" into [0, 1] output space
3. Choose a method of optimization
   - ✓ Gradient descent
4. Train your system and evaluate performance
   - Ideal method: blind testing of a trained classifier on new data
   - Other method: cross-validation
   - ✓ As before, we'll train on all data and won't evaluate performance via either method.
     - ✓ But beware this puts us in danger of over-fitting!!

Source: Ajay Jain

Logistic regression: classification problems

- Goal: Unlike linear regression, we want to predict discrete variables, e.g., $y^{(i)}\ in\ \{0, 1\}$
  - For now, we'll focus on binary classification only
    - E.g., tumor is/not malignant → $y = \{\ 0{:}malignant, 1{:}not\}$
    - Inputs x may be x_1 = size, x_2 = eccentricity, etc.

  - What should be the range of $h_\theta(x)$?
  - Why not just use linear regression for this?

Source: UFLDL Tutorial, Andrew Ng (Stanford)

Logistic regression: classification problems

- Specifically, we want to learn a function of the form:

$$P(y = 1|x) = h_\theta(x) = \frac{1}{1 + \exp(-\theta^\top x)} \equiv \sigma(\theta^\top x),$$
$$P(y = 0|x) = 1 - P(y = 1|x) = 1 - h_\theta(x).$$

Where: $\theta^T x = \theta_0 + \sum_{j=1}^{n} \theta_j x_j$

- $\sigma(\bullet)$ above is the "logistic" or "sigmoid" function

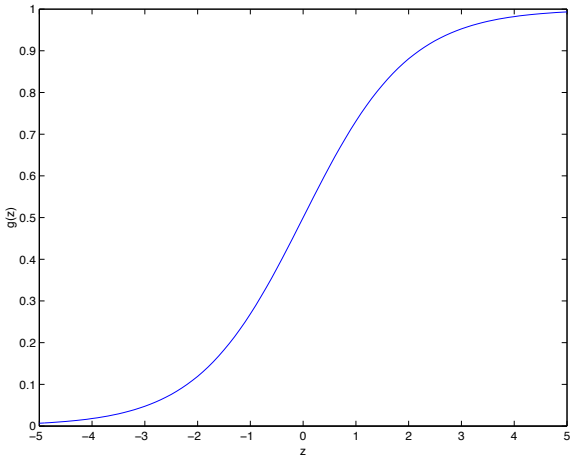
$P(y = 1\|x)$	probability $y = 1$ given x
$(x^{(i)},y^{(i)})$	i^{th} training example
θ^T	transpose(θ) = θ'
n	number of training examples

Source: UFLDL Tutorial, Andrew Ng (Stanford)

Logistic regression – why logistic?

- Why use a sigmoid function?

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

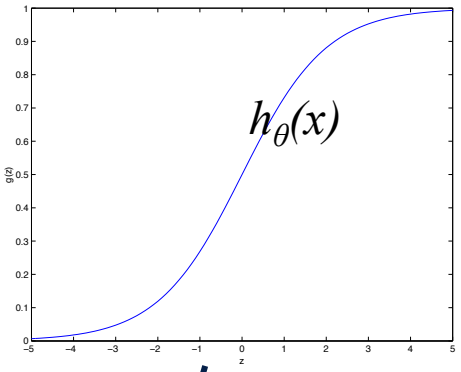


- This function "squashes" $\theta^T = \theta_0(x_0) + \theta_1 x_1 + \theta_2 x_2$ ... into the range *[0, 1]*, which may be used as a probability of assignment to the categories.
- Other smooth functions in this range could be used, but this has some nice properties (e.g., its derivative & use in generalized linear models).

Source: UFLDL Tutorial & CS229 Notes, Andrew Ng (Stanford)

Logistic regression cost function

- So we want to minimize some cost function $J(\theta)$ such that:
  - $P(y=1|x) = h_\theta(x)$ is large when x belongs to the "*1*" class, and small when x belongs to the "*0*" class:



$$J(\theta) = -\sum_i \left(y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right)$$

Only one of these terms is nonzero per training variable $(x^{(i)}, y^{(i)})$

$(x^{(i)},y^{(i)})$	i^{th} training example
θ^T	transpose(θ) = θ'
n	number of training examples

Where: $h_\theta(x) = \dfrac{1}{1 + \exp(-\theta^\top x)}$

$\theta^T x = \theta_0 + \sum_{j=1}^{n} \theta_j x_j$

Source: UFLDL Tutorial & CS229 Notes, Andrew Ng (Stanford)

UCSF

Logistic regression gradient

- As with linear regression, we need to take the derivative of the cost function:

$$J(\theta) = -\sum_i \left(y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right)$$

Where: $h_\theta(x) = \dfrac{1}{1 + \exp(-\theta^\top x)}$

- This is:

θ^T	transpose(θ) = θ'

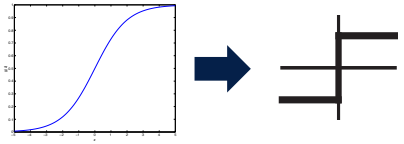
$$\nabla_\theta J(\theta) = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_1} \\ \frac{\partial J(\theta)}{\partial \theta_2} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{bmatrix} \Rightarrow \frac{\partial J(\theta)}{\partial \theta_j} = \sum_i x_j^{(i)} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

- This looks a lot like the gradient for linear regression!
  - But now $h_\theta(x)$ is nonlinear

j	denotes input variable
$(x^{(i)},y^{(i)})$	i^{th} training example

Source: UFLDL Tutorial, Andrew Ng (Stanford)

Perceptron via logistic regression



- What if you "forced" logistic regression's output to values of either *0* or *1*:

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

$$h_\theta(x) = g(\theta^T x)$$

Recall that for logistic regression:

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

- …and then used a stochastic gradient descent update rule:

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$$

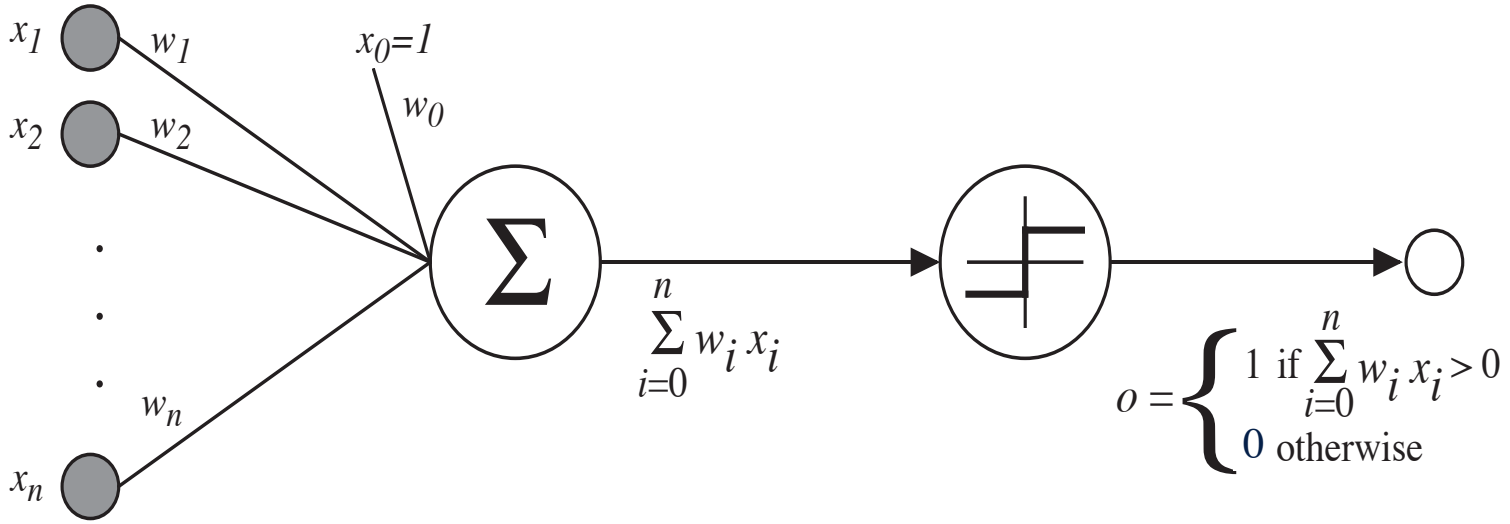
- This is the perceptron learning algorithm.
  - And it's *not* logistic regression anymore!

Recall:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \sum_i x_j^{(i)} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

Source: CS229 Notes, Andrew Ng (Stanford)

Perceptron: a linear function



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

The perceptron encodes a line in a high-dimensional space. For machine learning, it can solve simple problems: those that are linearly separable.

Note on syntax
$w_i \rightarrow \theta_i$
$o(x) \rightarrow h_\theta(x)$

Source: Ajay Jain (adapted)

UCSF

The perceptron training rule is just gradient descent

Goal: Minimize the error over the training examples

$$o = w_0 + w_1 x_1 + \cdots + w_n x_n$$

$$E(\bar{w}) = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

Cost function

w_i	→	θ_i
$o(x)$	→	$h_\theta(x)$
E	→	$J(\theta)$, the cost function
t_d	→	y_d, the "target" output
D		set of training examples $\{(x_i, y_i), \ldots\}$

Source: Ajay Jain (adapted)

Perceptron training gradient

The gradient is then defined by:

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\left(\frac{1}{2}\sum_d (t_d - o_d)^2\right)$$

$$= \sum_d (t_d - o_d)(-x_{i,d})$$

Does this training rule make sense?

- When we have the right answer, $t_d - o_d = 0$ → no change to weights.
- When an input $x_{i,d}$ is zero → no change to weights.
- If output is too low (and $x_{i,d}$ is positive), then error gradient is negative, and contribution to weight is positive.

Source: Ajay Jain (adapted)

Perceptron training algorithm

- Goal: Minimize $J(\theta)$ over choices of θ
  - Given a set of i training examples $\{(x_1, y_1), \ldots, (x_i, y_i)\}$

1. Initialize weights $\theta = [\theta_0, \theta_1, \ldots, \theta_j] =$ *vector of small random values*
2. Define a cost function, $J(\theta)$
3. Define a gradient, $\nabla_\theta J(\theta)$
4. For $t = 1, \ldots, T$:
   - Set $\theta = \theta + \alpha\,(y - h_\theta(x))\,x$
   - Halt on convergence and report θ

Expanded, this would be:

For all training examples i

For all weights j

$$\theta_j = \theta_j + \alpha\,(y_i - h_\theta(x_i))\,x_i$$

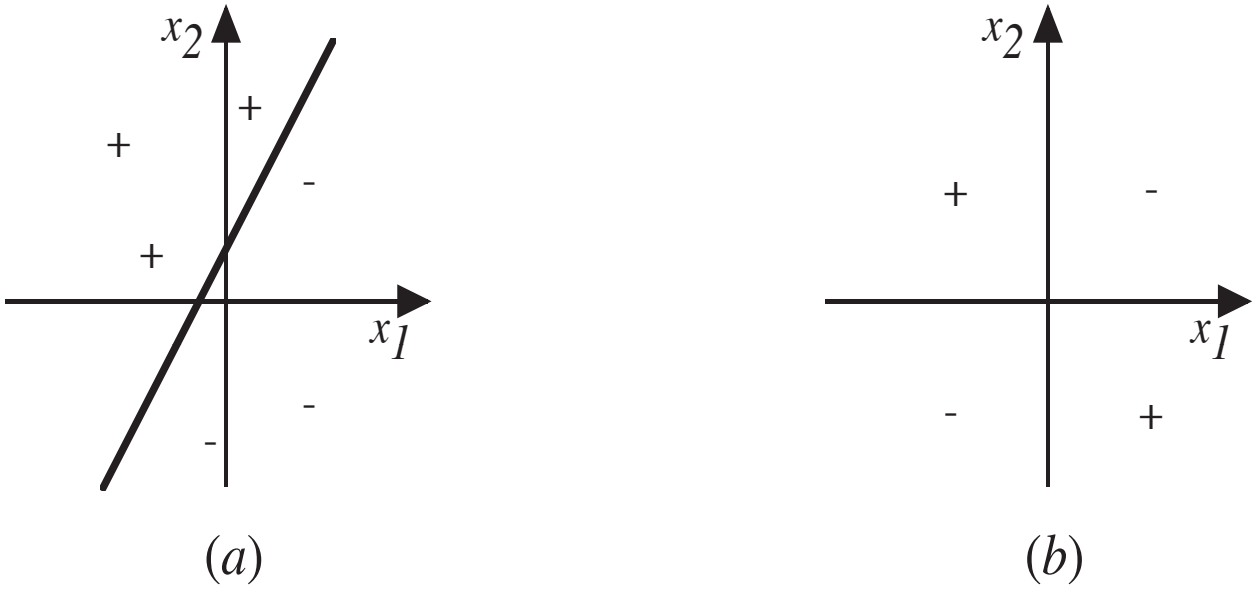
Perceptron algorithm properties

- The perceptron training algorithm is ***guaranteed*** to succeed if
  - Training examples are linearly separable
  - Learning rate is sufficiently small

- Notes
  - Multiple linear units are always representable as a single linear unit
  - ***Very few*** problems are linearly separable
  - Can perform incremental gradient descent (modify weights after each example): sometimes converges faster, allows for skipping examples that are “close enough”

Source: Ajay Jain (adapted)

The decision surface of a perceptron limits it to linearly separable problems



Minsky and Papert killed “neural nets” in 1969 for about 15 years by showing that perceptrons couldn’t solve simple problems like *(b)*.

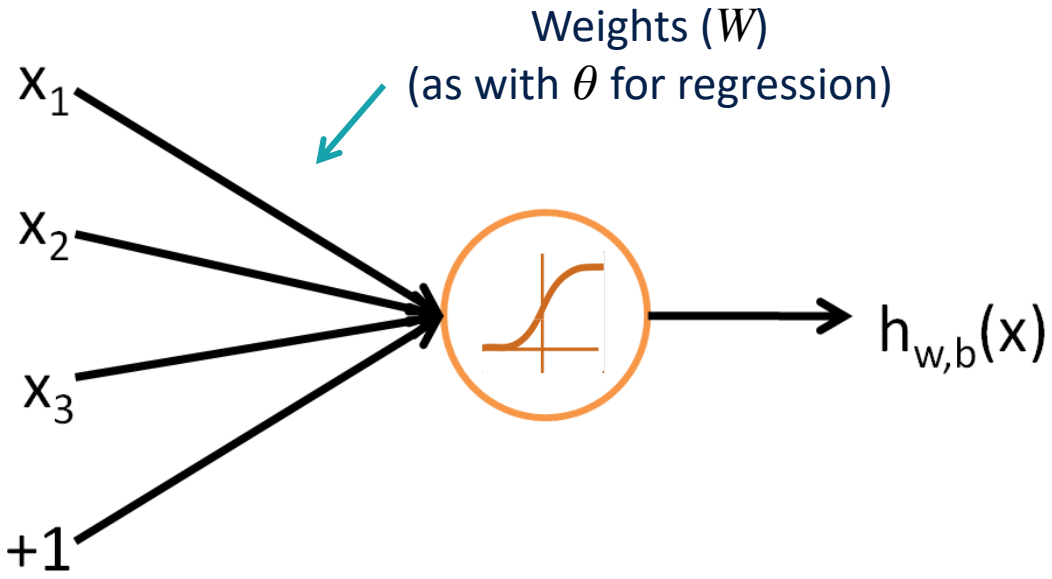
Source: Ajay Jain

A brief history

- 1795: Gauss proposed least squares (astronomy)
- 1940s: logistic regression (statistics)
- 1952: Arthur Samuel built program that learned to play checkers (AI)
- 1957: Rosenblatt invented Perceptron algorithm
- 1969: Minsky and Papert “killed” machine learning
- 1980s: neural networks (backpropagation, from 1960s)
- 1990: interface with optimization/statistics, SVMs
- 2000s: structured prediction, revival of neural networks
- 2006-2012 onward: deep learning

Shamelessly lifted from: CS221 AI Principles & Techniques, Percy Liang (Stanford), lecture 3, slide 81

Simplest neural net: a "neuron"



$$h_{W,b}(x) = f(W^T x) = f(\sum_{i=1}^{3} W_i x_i + b)$$

f is the "activation function": $f(z) = \frac{1}{1 + \exp(-z)}$

Source: UFLDL Tutorial, Andrew Ng (Stanford)

Activation functions

- Using a sigmoid activation function, we match the input-output mapping for logistic regression:

$$h_\theta(x) = \frac{1}{1+\exp(-\theta^\top x)}$$

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

$$f'(z) = f(z)(1-f(z))$$

- Other common choices are *tanh*

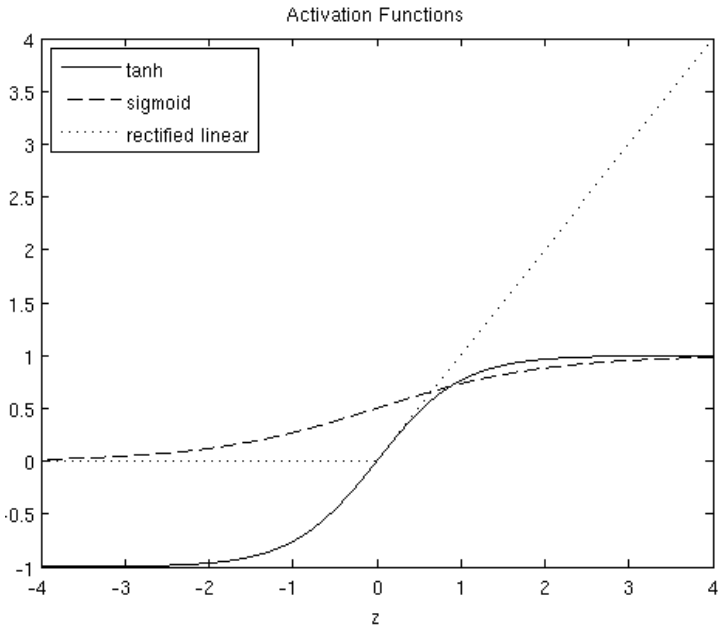
$$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$f'(z) = 1 - (f(z))^2$$

- And the rectified linear function

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

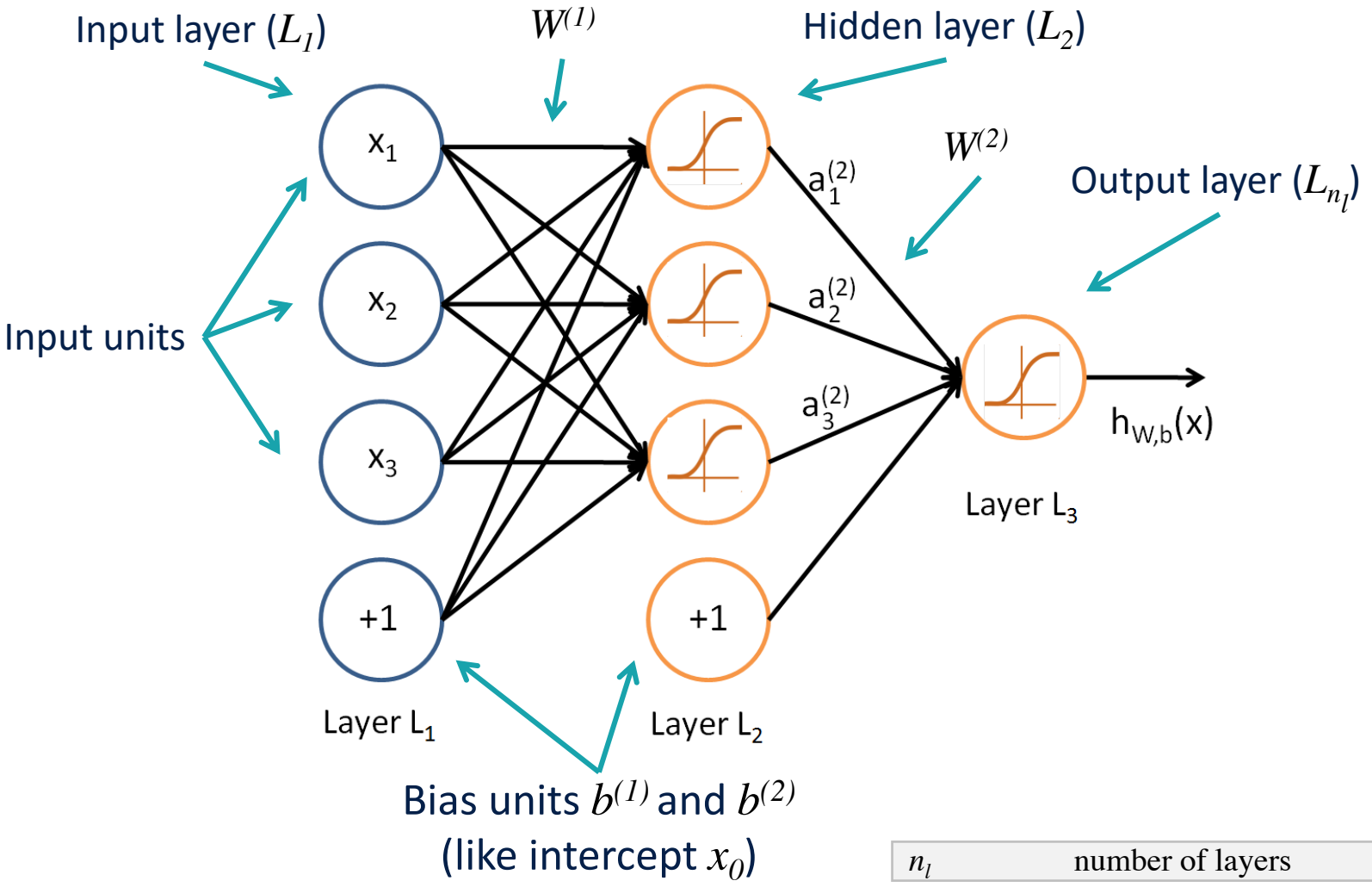
$$f'(z) = 0 \text{ if } z \leq 0, \text{ else } 1$$



Note the different output ranges (y-axis)

Source: UFLDL Tutorial, Andrew Ng (Stanford)

Neural network terminology



n_l	number of layers

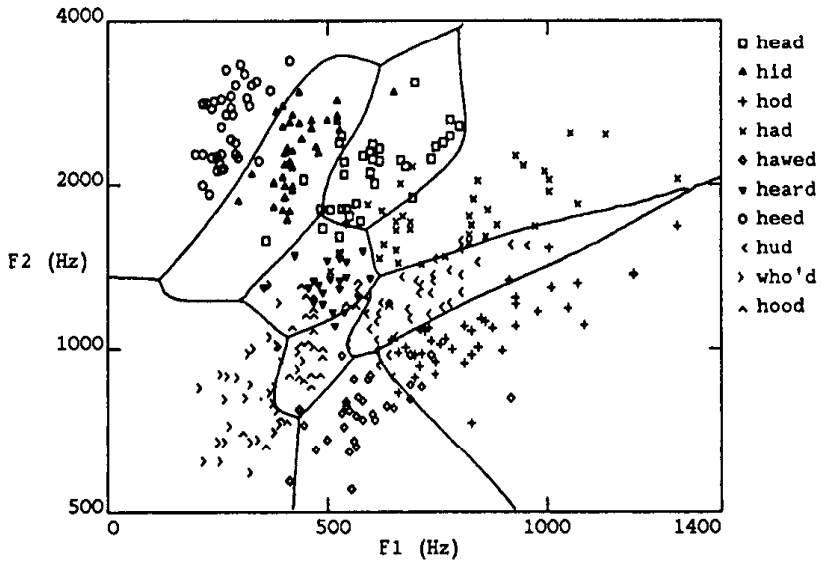
Source: UFLDL Tutorial, Andrew Ng (Stanford)

UCSF

What's the point of hidden layers?

- Think of intermediate hidden units as learned high-level features.
  - Then a neural network can be viewed as learning the features for, say, a linear classifier.

- Functional complexity by combining non-linear activation functions.
- This can be useful for dealing with non-linear issues in your features such as:
  - Non-monotonicity
  - Saturation
  - Interactions between features



More info: CS221 AI Principles & Techniques, Percy Liang (Stanford); figure from Ajay Jain