Statistical Machine Learning Methods for Bioinformatics

III. Neural Network Theory

Jianlin Cheng, PhD

Department of Computer Science

University of Missouri

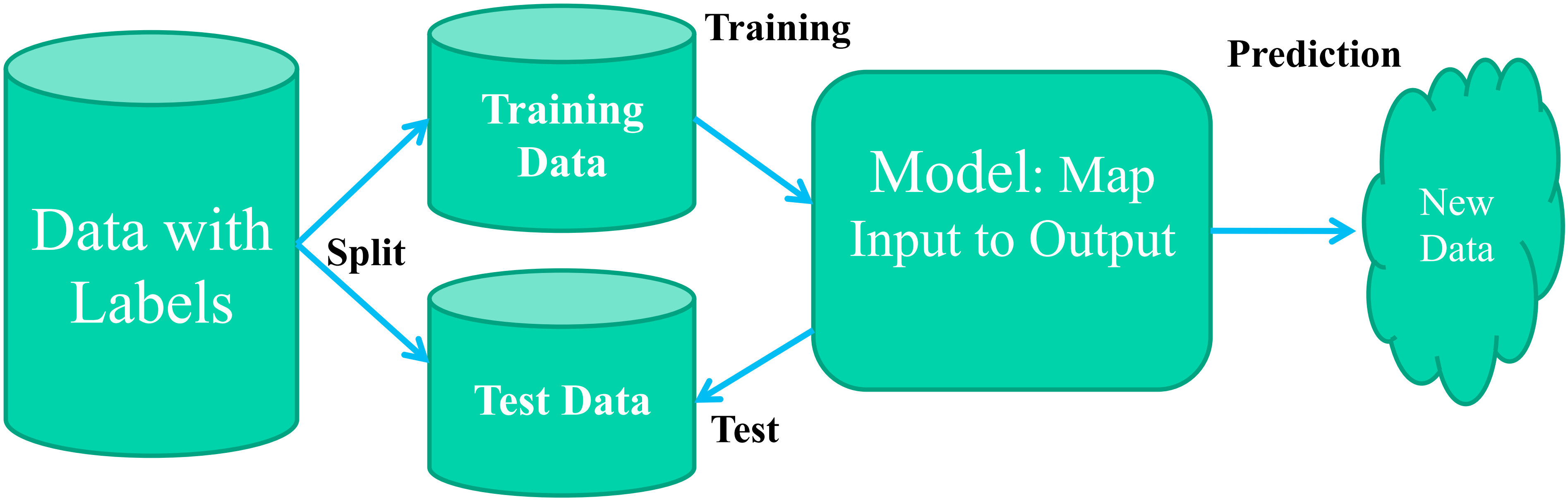
2012

Classification Problem

Input								Output
Legs	weight	size	….				Feature m	Category / Label
4	100							Mammal
80	0.1							Bug

Question: How to automatically predict output given input?
Idea: Learn from known examples and generalize to unknown ones.

Data Driven Machine Learning Approach



Input: words of news
Output: politics, sports, entertainment, …

Training: Build a model (classifier)
Test: Test the model

Key idea: Learn from known data and Generalize to unseen data

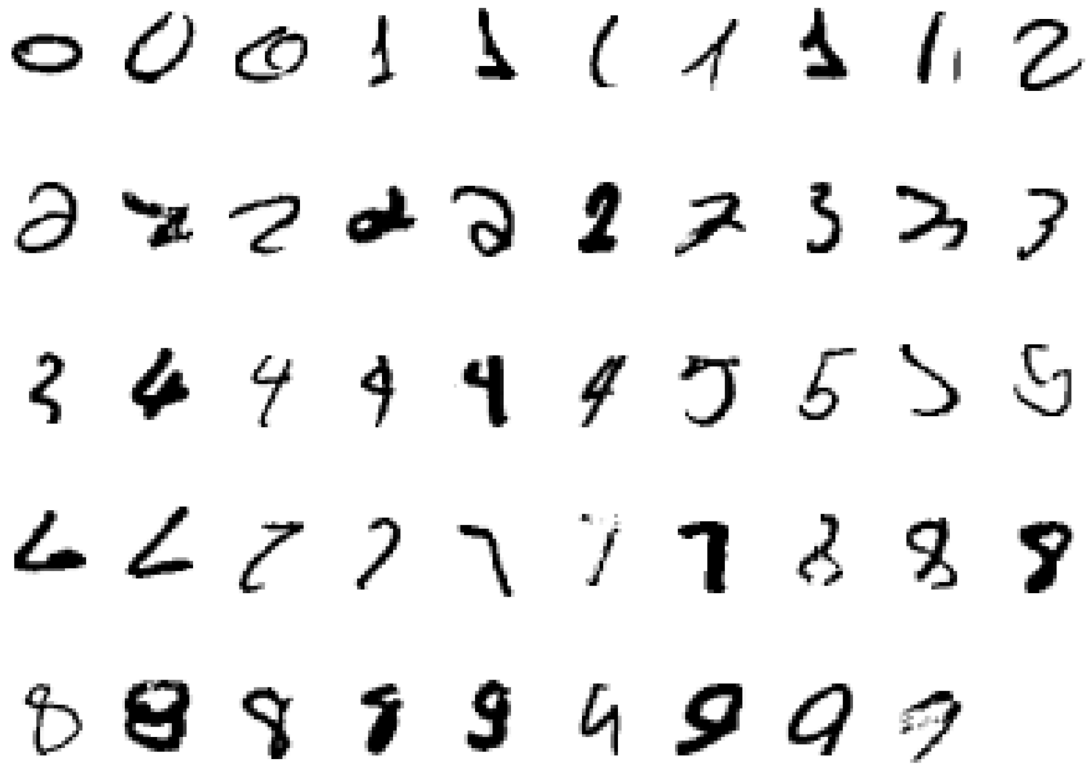
Outline

- Introduction
- Linear regression
- Linear Discriminant function (classification)
- One layer neural network / perceptron
- Multi-layer network
- Recurrent neural network
- Prevent overfitting
- Speedup learning

Machine Learning

- **Supervised learning** (training with labeled data), **un-supervised learning** (clustering un-labeled data), and **semi-supervised learning** (use both labeled and unlabeled data)
- Supervised learning: **classification** and **regression**
- **Classification**: output is discrete value
- **Regression**: output is real value

Learning Example: Recognize Handwriting



Classification: recognize each number
Clustering: cluster the same numbers together
Regression: predict the index of Dow-Jones

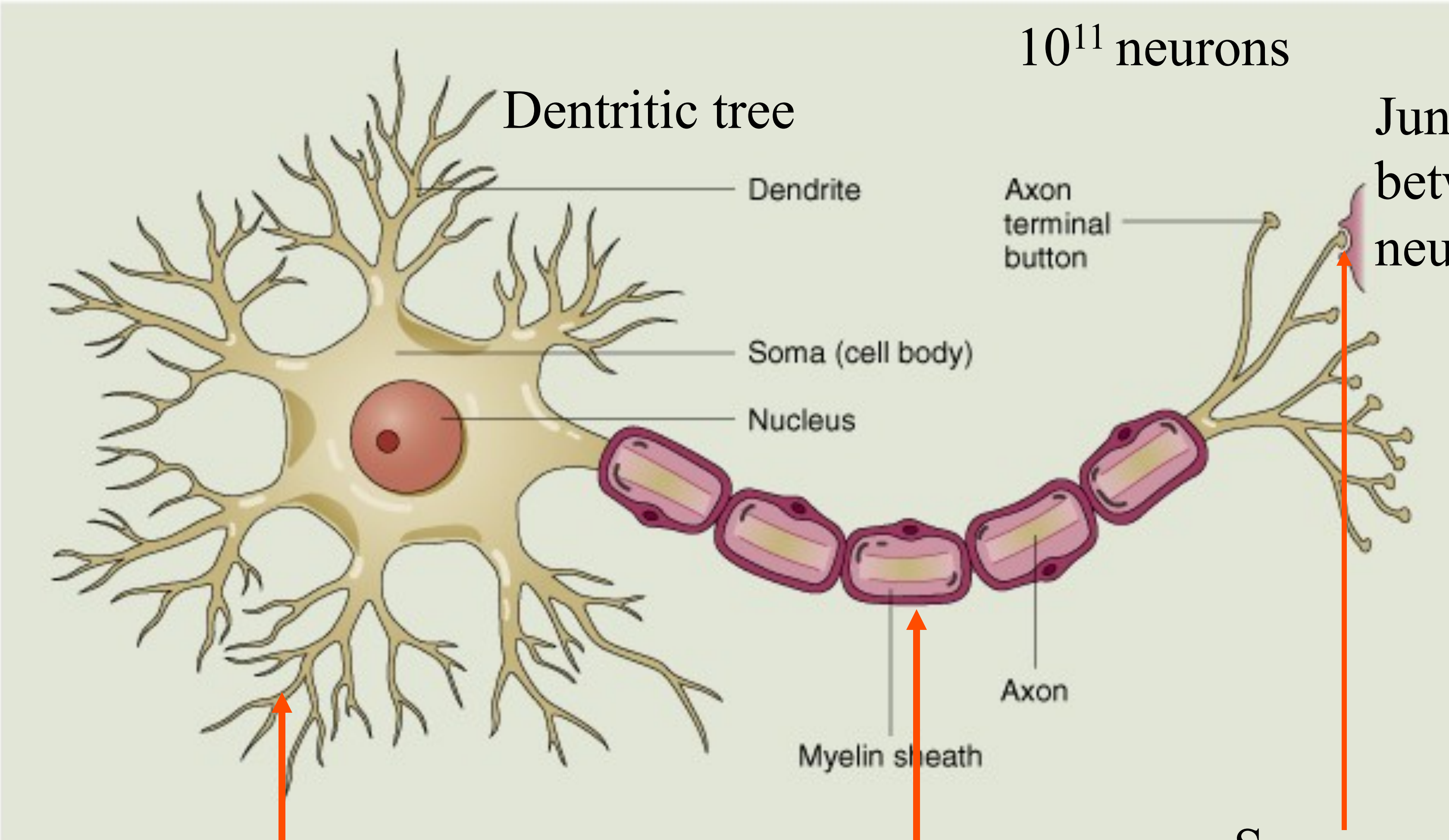
Neural Network

- Neural Network can do both supervised learning and un-supervised learning
- Neural Network can do both regression and classification
- Neural Network has both statistical and artificial intelligence roots

Roots of Neural Network

- Artificial intelligence root (neuron science)
- **Statistical root** (linear regression, generalized linear regression, discriminant analysis. This is our focus.)

A Typical Cortical Neuron

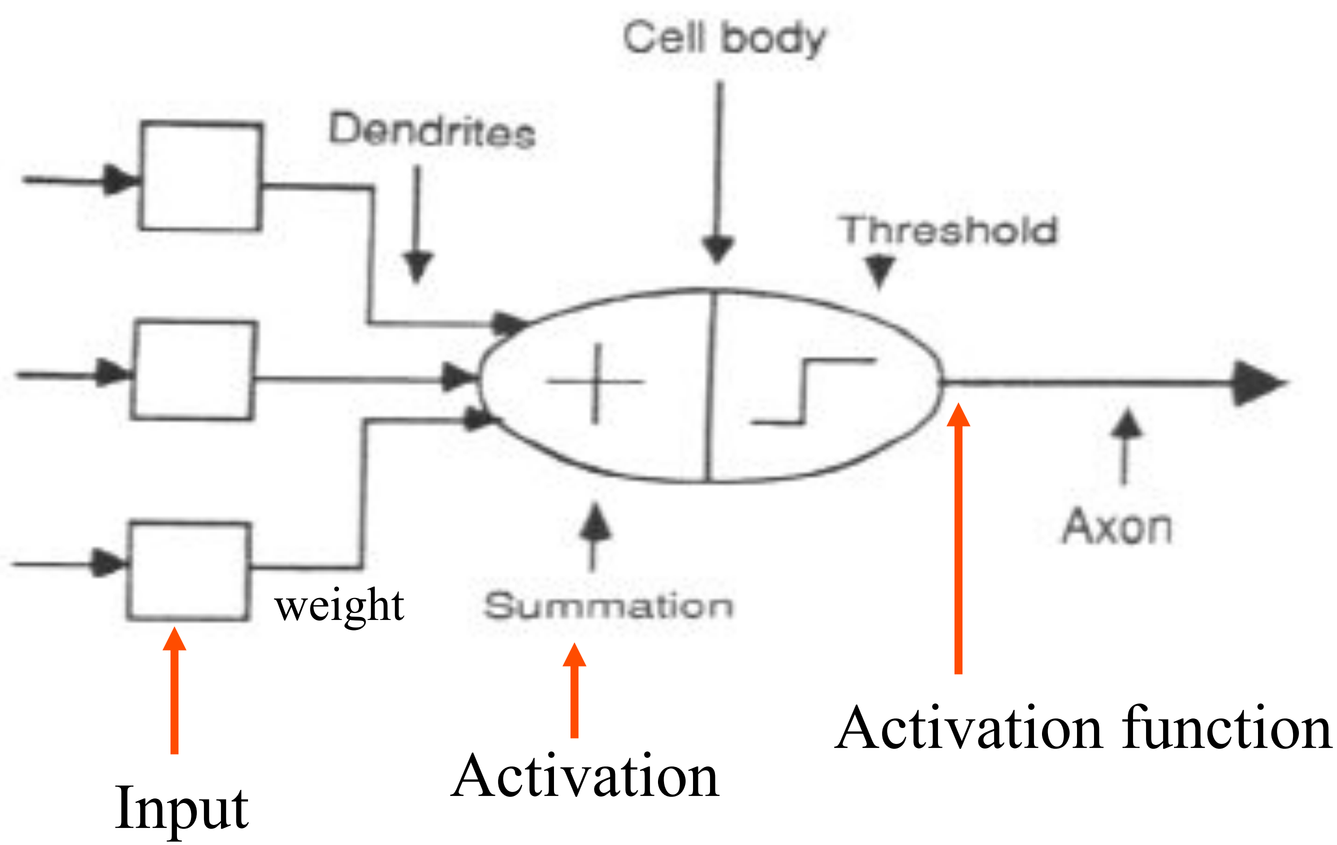


Synapse: control release chemical transmitters.

Axon: generate Potentials (Fire/not Fire)

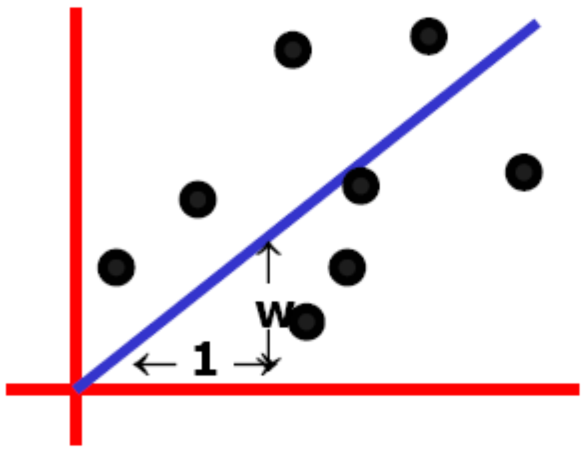
Collect chemical signals

A Neural Model



Adapted from http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html

Statistics Root: Linear Regression Example



DATASET

inputs	outputs
$x_1 = 1$	$y_1 = 1$
$x_2 = 3$	$y_2 = 2.2$
$x_3 = 2$	$y_3 = 2$
$x_4 = 1.5$	$y_4 = 1.9$
$x_5 = 4$	$y_5 = 3.1$

Fish length vs. weight?

X: input or predictor
Y: output or response
Goal: learn a linear function E[y|x] = wx + b.

Adapted from A. Moore, 2003

Linear Regression

Definition of a linear model:

- $y = wx + b + noise$.
- noise ~ N $(0, \sigma^2)$, assume σ is a constant.
- $y \sim N(wx + b, \sigma^2)$
- Estimate expected value of y given x ($E[y|x] = wx + b$) .
- Given a set of data $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, to find the optimal parameters w and b.

Objective Function

- Least square error: $\sum_{i=1}^{N}(y_i - wx_i - b)^2$
- Maximum Likelihood: $\prod_{i=1}^{N} P(y_i \mid x_i, w, b)$
- Minimizing square error is equivalent to maximizing likelihood

Maximize Likelihood

$$\prod_{i=1}^{N} P(y_i \mid x_i, w, b) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i - wx_i - b)^2}{2\sigma^2}}$$

Minimize negative log-likelihood:

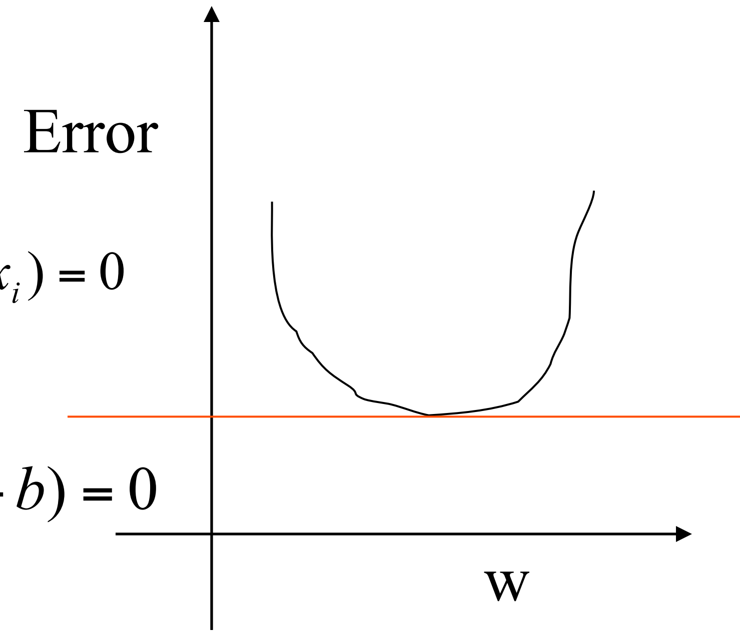
$$-\log\left(\prod_{i=1}^{N} P(y_i \mid x_i, w, b)\right) = -\log\left(\prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y_i - wx_i - b)^2}{2\sigma^2}}\right) = -\sum_{i=1}^{N}\left(-\log(\sqrt{2\pi\sigma^2}) - \frac{(y_i - wx_i - b)^2}{2\sigma^2}\right)$$

$$= \sum_{i=1}^{N}\left(\log(\sqrt{2\pi\sigma^2}) + \frac{(y_i - wx_i - b)^2}{2\sigma^2}\right)$$

Note: σ is a constant.

1-Variable Linear Regression

Minimize E $= \displaystyle\sum_{i=1}^{N} (y_i - wx_i - b)^2$



$$\frac{\partial E}{\partial W} = \sum_{i=1}^{N} 2(y_i - wx_i - b) * (-x_i) = \sum_{i=1}^{N} 2(-y_i x_i + wx_i^2 + bx_i) = 0$$

$$\frac{\partial E}{\partial b} = \sum_{i=1}^{N} 2(y_i - wx_i - b) * (-1) = \sum_{i=1}^{N} 2(-y_i + wx_i + b) = 0$$

$$w = \frac{\sum_{i=1}^{N} x_i y_i - N\overline{x}\overline{y}}{\sum_{i=1}^{N} x_i^2 - N\overline{x}\overline{x}} \qquad b = \frac{\sum_{i=1}^{N} (y_i - wx_i)}{N}$$

Linear Regression Demo

http://www.calpoly.edu/~srein/StatDemo/All.html

Multivariate Linear Regression

- How about multiple predictors: (x_1, x_2, …, x_d).
- $y = w_0 + w_1 x_1 + w_2 x_2 + \ldots + w_d x_d + \varepsilon$
- For multiple data points, each data point is represented as (y_i, x_i), x_i consists of d predictors (x_{i1}, x_{i2}, …, x_{id}).
- $y_i = w_0 + w_1 x_{i1} + w_2 x_{i2} + \ldots + w_d x_{id} + \varepsilon$

A Motivating Example

- Each day you get lunch at the cafeteria.
  - Your diet consists of fish, chips, and beer.
  - You get several portions of each
- The cashier only tells you the total price of the meal
  - After several days, you should be able to figure out the price of each portion.
- Each meal price gives a linear constraint on the prices of the portions:

$$price = x_{fish} w_{fish} + x_{chips} w_{chips} + x_{beer} w_{beer}$$

G. Hinton, 2006

Matrix Representation

***n* data points, *d* dimension**

$$\begin{pmatrix} y_1 \\ y_2 \\ \cdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1d} \\ 1 & x_{21} & \cdots & x_{2d} \\ \cdots & . & \cdots & . \\ 1 & x_{n1} & \cdots & x_{nd} \end{pmatrix} \times \begin{pmatrix} w_0 \\ w_1 \\ \cdots \\ w_d \end{pmatrix} + \varepsilon$$

n*1 n*(d+1) (d+1)*1

Matrix Representation: $\mathbf{Y} = \mathbf{XW} + \boldsymbol{\varepsilon}$

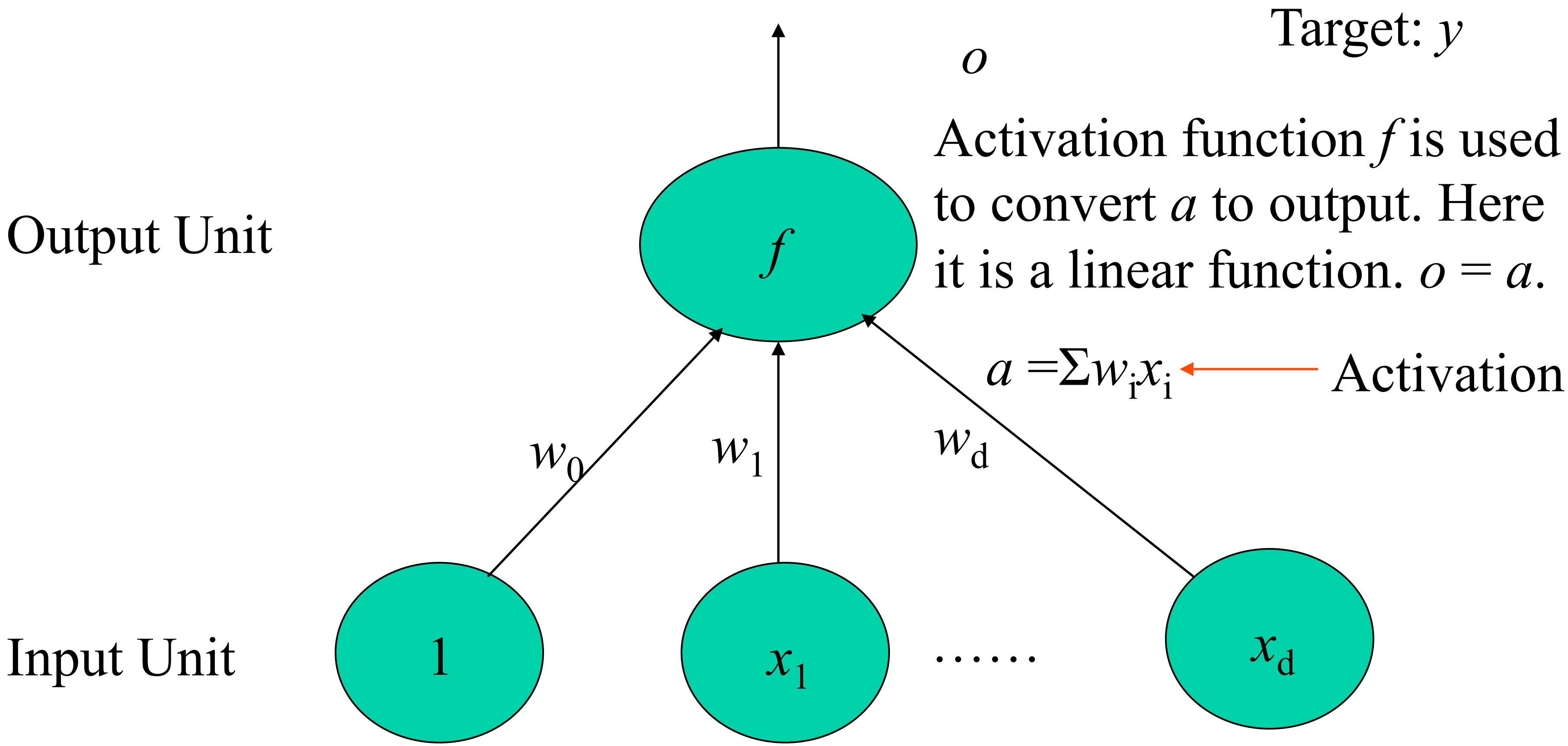
Multivariate Linear Regression

- Goal: minimize square error = $(Y-XW)^T(Y-XW) = Y^TY - 2X^TWY + W^TX^TXW$
- Derivative: $-2X^TY + 2X^TXW = 0$
- $W = (X^TX)^{-1}X^TY$
- Thus, we can solve linear regression using matrix inversion, transpose, and multiplication.

Difficulty and Generalization

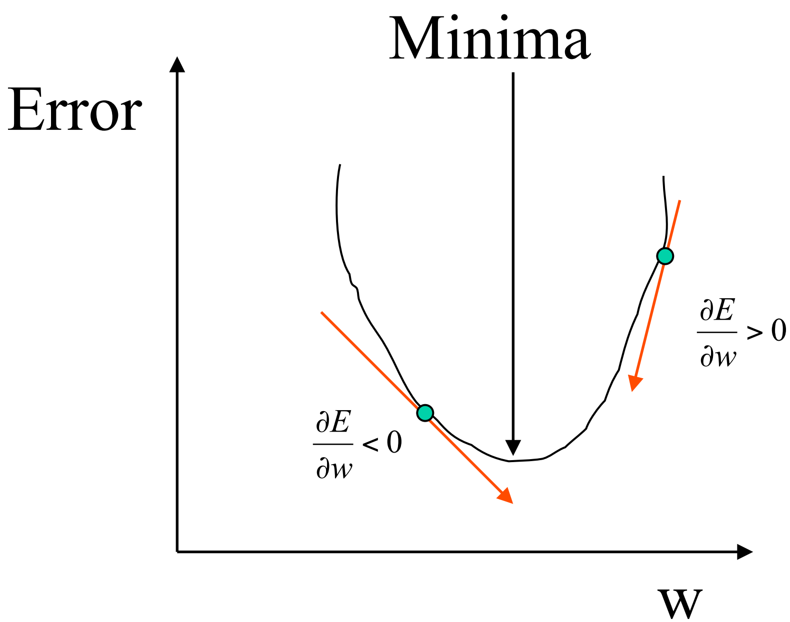
- Numerical computation issue. (a lot data points. Matrix inversion is impossible.)
- Singular matrix (determinant is zero) : no inversion
- How to handle non-linear data?
- Turns out neural network and its iterative learning algorithm can address this problem.

Graphical Representation: One Layer Neural Network for Regression



Gradient Descent Algorithm

- For a data $x = (x_1, x_2, \ldots x_d)$, error $E = (y - o)^2 = (y - w_0 x_0 - w_1 x_1 - \ldots - w_d x_d)^2$
- Partial derivative: $\nabla E|_{w_i} = \frac{\partial E}{\partial w_i} = 2(y-o)\frac{\partial o}{\partial w_i} = 2(y-o)(-x_i) = -2(y-o)x_i$



Update rule:

$$w_i^{(t+1)} = w_i^{(t)} + \eta(y - o)x_i$$

Famous Delta Rule

Algorithm of One-Layer Regression Neural Network

- Initialize weights w (small random numbers)
- **Repeat**

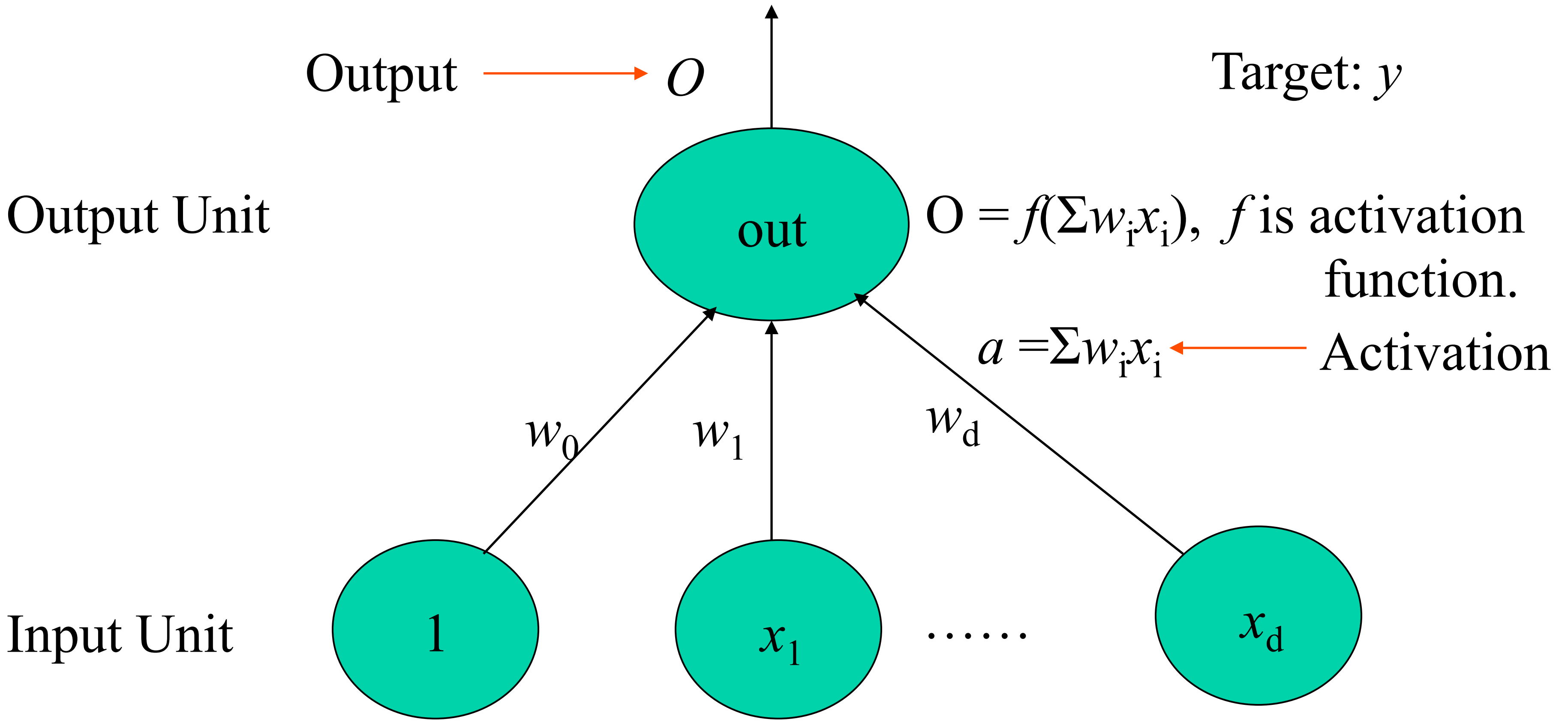
  Present a data point $x = (x_1, x_2, \ldots, x_d)$ to the network and compute output o.

  if $y > o$, add ηx_i to w_i.

  if $y < o$, add $-\eta x_i$ to w_i.
- **Until** $\Sigma(y_k - o_k)^2$ is zero or below a threshold or reaches the predefined number of iterations.

Comments: online learning: update weight for every x. batch learning: update weight every batch of x (i.e. $\Sigma \eta x_i$).

Graphical Representation: One Layer Neural Network for Regression

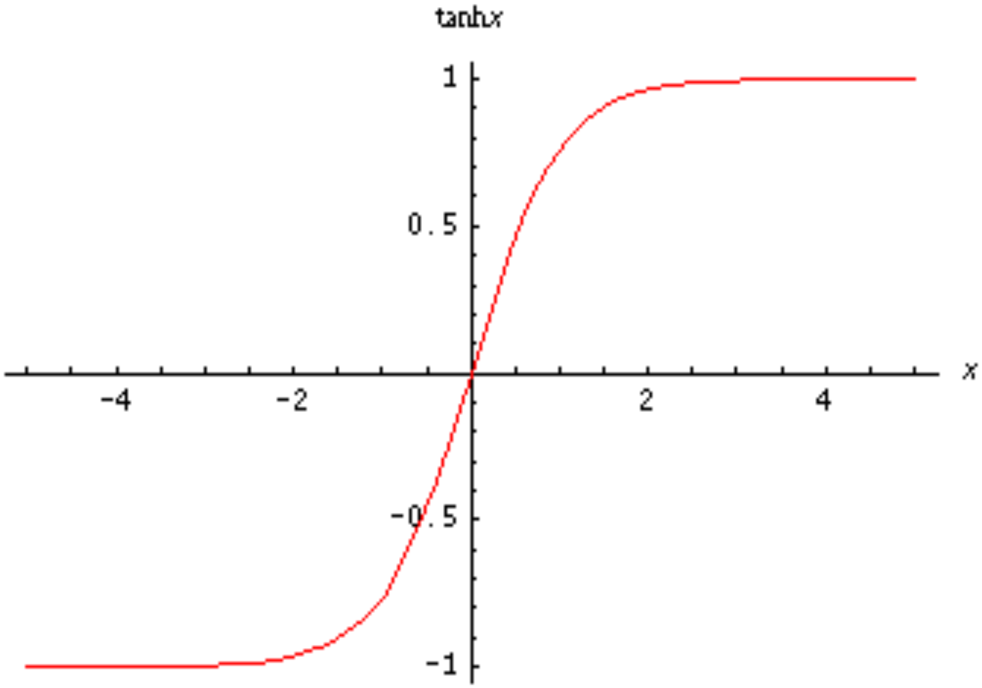


What about Hyperbolic Tanh Function for Output Unit

- Can we use activation function other than linear function?
- For instance, if we want to limit the output to be in [-1, +1], we can use hyperbolic Tanh function:

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

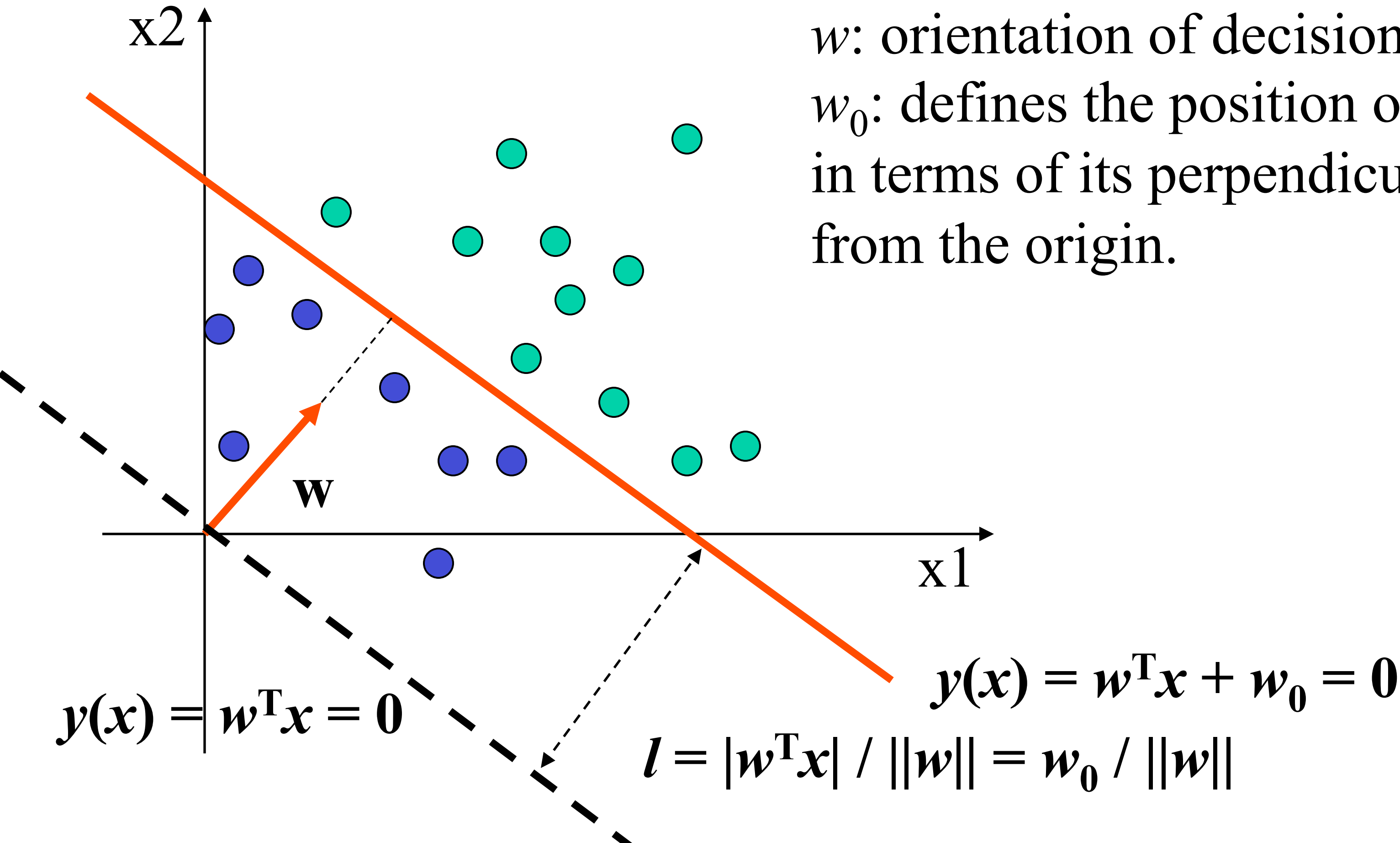
- The only thing to change is to use the new gradient.



Two-Category Classification

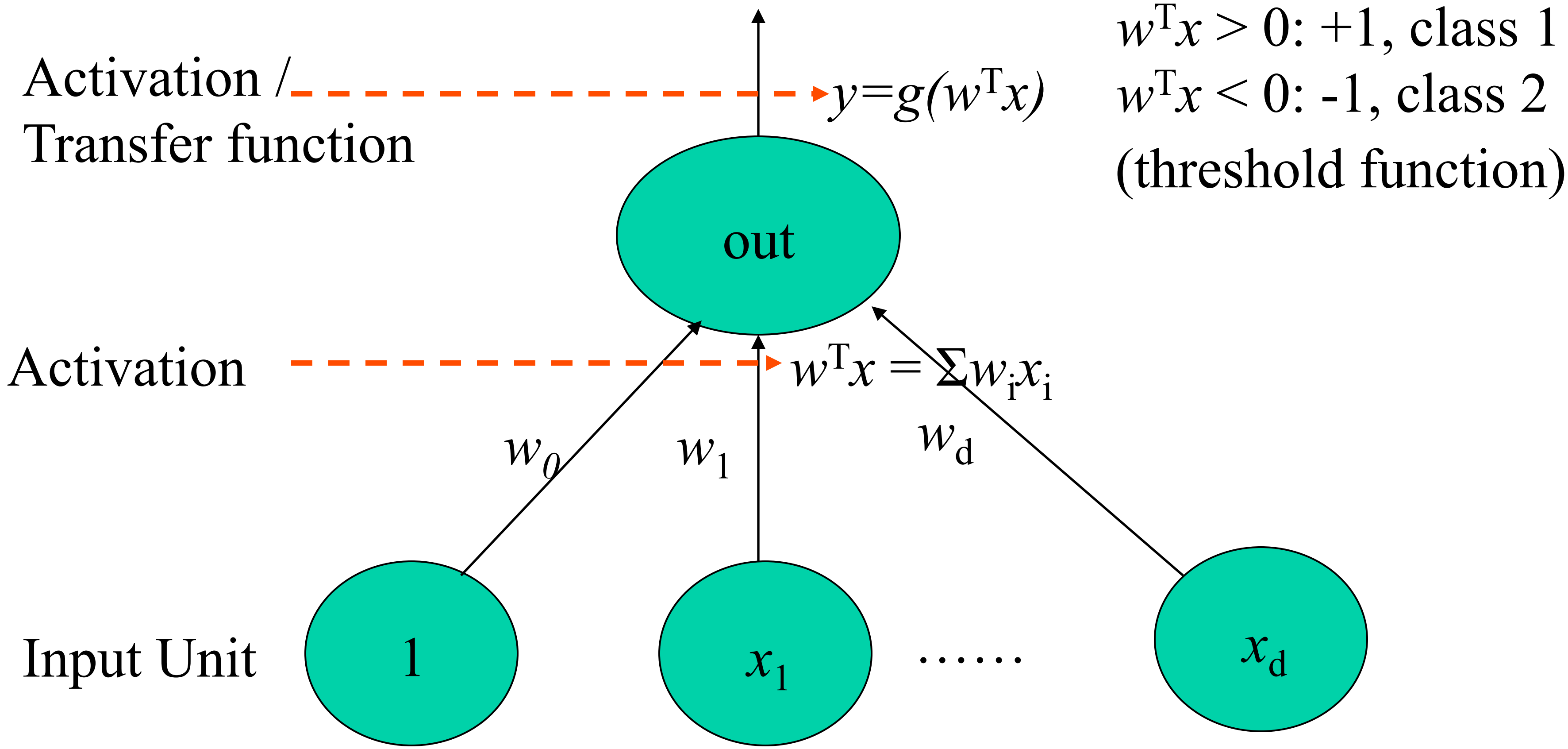
- Two classes: C_1 and C_2.
- Input feature vector: x.
- Define a discriminant function $y(x)$ such that x is assigned to C_1 if $y(x) > 0$ and to class C_2 if $y(x) < 0$.
- Linear discriminant function: $y(x) = w^T x + w_0 = \bar{w}^T \bar{x}$, where $\bar{x} = (1, x)$.
- w: weight vector, w_0: bias.

A Linear Decision Boundary in 2-D Input Space



w: orientation of decision boundary
w_0: defines the position of the plan in terms of its perpendicular distance from the origin.

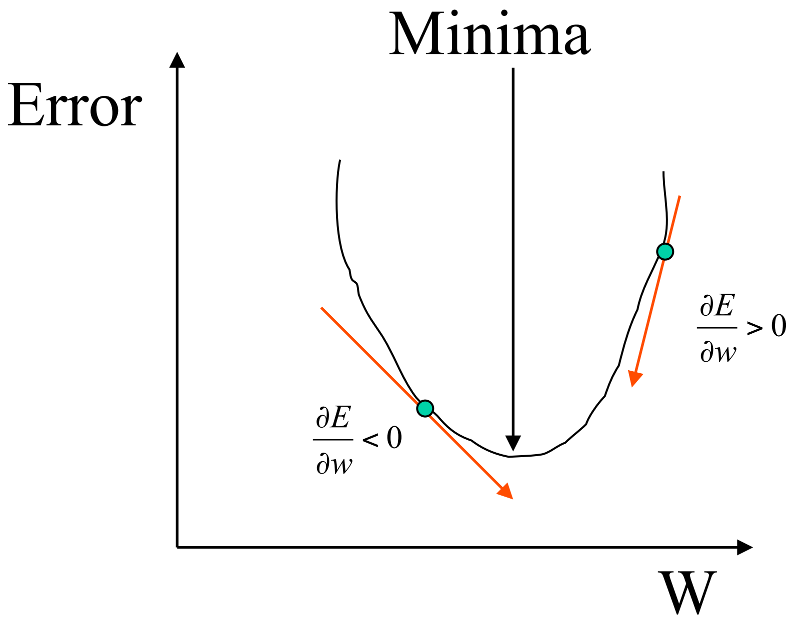
Graphical Representation: Perceptron, One-Layer Classification Neural Network



Perceptron Criterion

- Minimize classification error
- Input data (vector): $x^1, x^2, \ldots, x^N$ and corresponding target value $t^1, t^2, \ldots, t^N$.
- Goal: for all x in C_1 ($t = 1$), $w^T x > 0$, for all x in C_2 ($t = -1$), $w^T x < 0$. Or for all x: $w^T x t > 0$.
- Error: $E^{perc}(w) = -\sum_{x^n \in M} w^T x^n t^n$. M is the set of misclassified data points.

Gradient Descent



For each misclassified data point, adjust weight as follows:

$$w = w - \frac{\partial E}{\partial w} \times \eta = w + \eta\, x^n t^n$$

Perceptron Algorithm

- Initialize weight w
- **Repeat**

  For each data point (x^n, t^n)

  Classify each data point using current w.

  If $w^T x^n t^n > 0$ (correct), do nothing

  If $w^T x^n t^n < 0$ (wrong), $w^{new} = w + \eta x^n t^n$

  $w = w^{new}$
- **Until** w is not changed (all the data will be separated correctly, if data is linearly separable) or error is below a threshold.

Rosenblatt, 1962

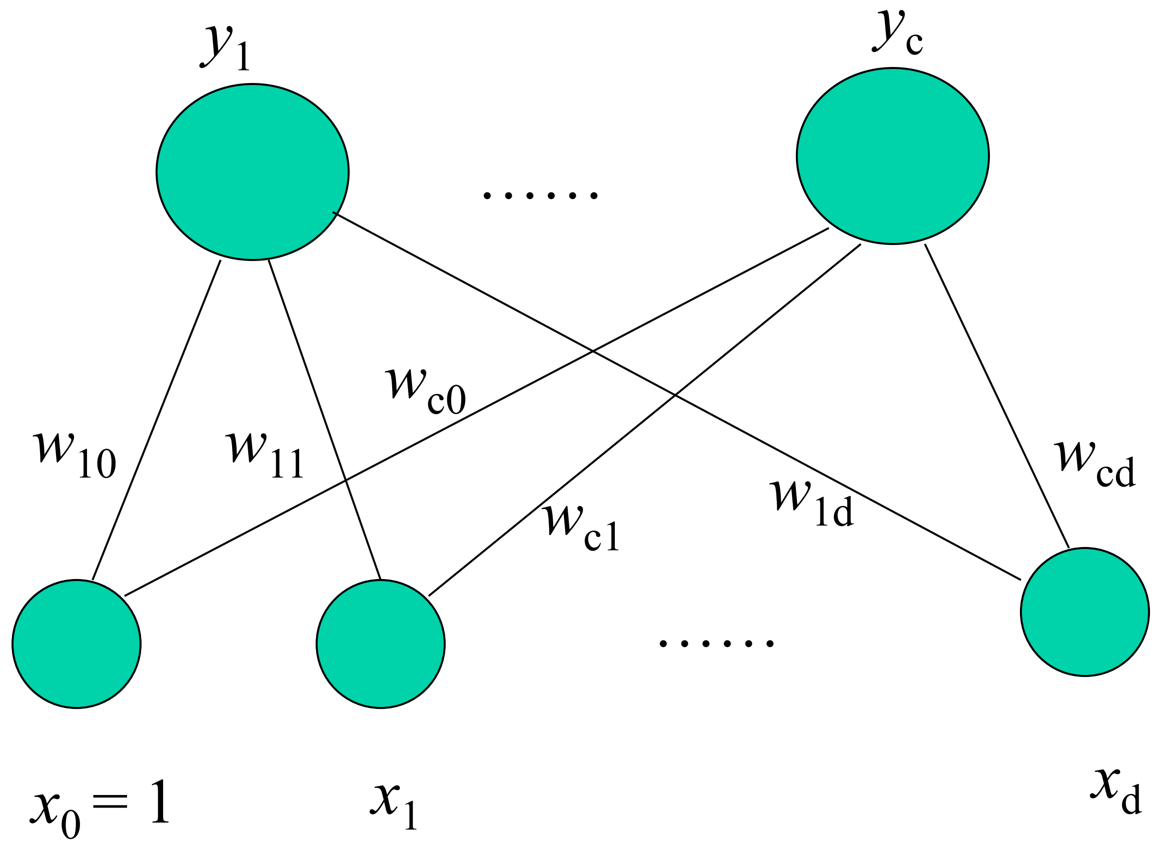
Perceptron Convergence Theorem

- For any data set which is linearly separable, the algorithm is guaranteed to find a solution in a finite number of steps (Rosenblatt, 1962; Block 1962; Nilsson, 1965; Minsky and Papert 1969; Duda and Hart, 1973; Hand, 1981; Arbib, 1987; Hertz et al., 1991)

Multi-Class Linear Discriminant Function

- c classes. Use one discriminant function $y_k(x) = w_k^T x + w_{k0}$ for each class C_k.
- A new data point x is assigned to class C_k if $y_k(x) > y_j(x)$ for all $j \neq k$.

One-Layer Multi-Class Perceptron



How to learn it?

Muti-Threshold Perceptron Algorithm

- Initialize weight w
- **Repeat**

  Present data point x to the network, if classification is correct, do nothing.

  if x is wrongly classified to C_i instead of true class C_j, adjust weights connected to C_i and C_j as follows.

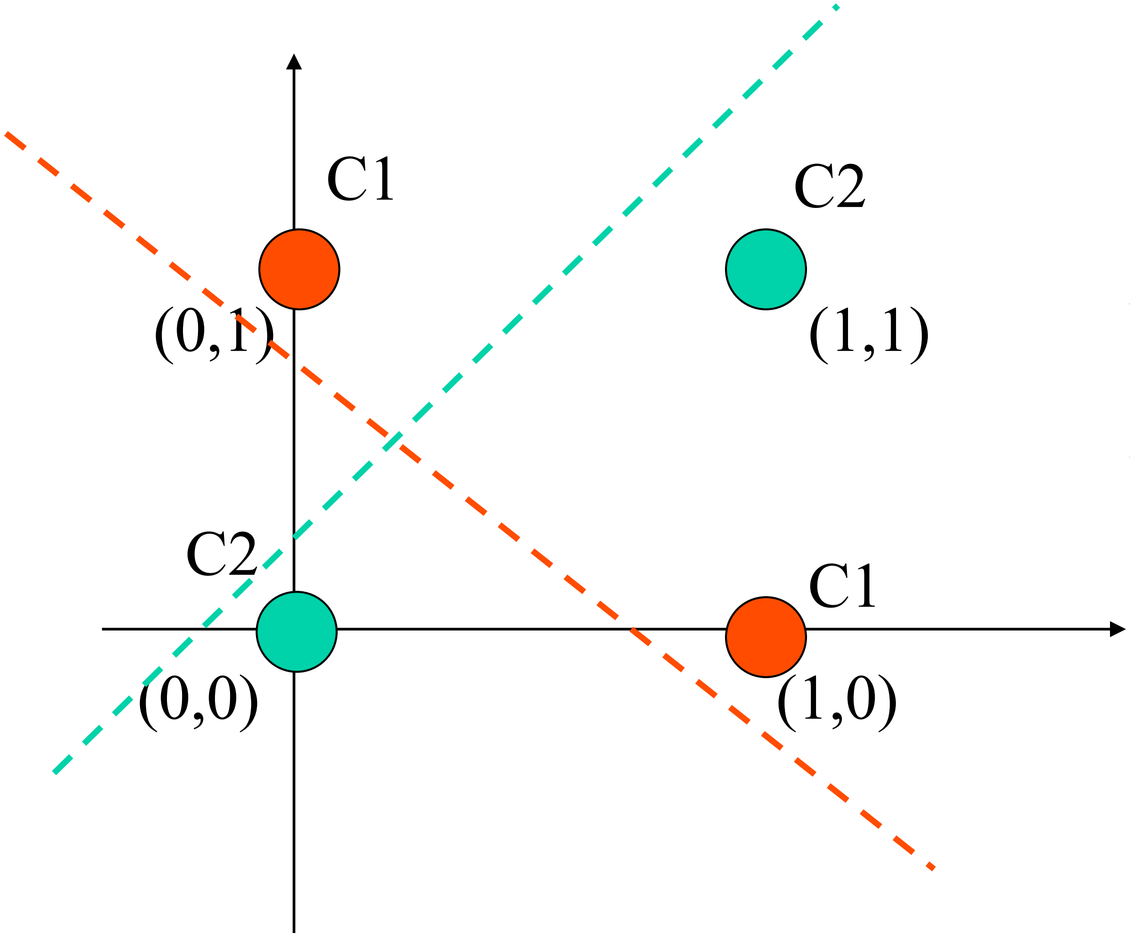
  Add $-\eta x_k$ to $w_{ik.}$ Add ηx_k to w_{jk}
- **Until** misclassification is zero or below a threshold.

Note: may also Add $-\eta x_k$ to w_{lk} for any l, $y_l > y_j$.

Limitation of the Perceptron

- Can’t not separate non-linear data completely.
- Or can’t not fit non-linear data well.
- Two directions to attack the problem: (1) extend to multi-layer neural network (2) map data into high dimension (SVM approach)

Exclusive OR Problem



Perceptron (or one-layer neural network) can not learn a function to separate the two classes perfectly.

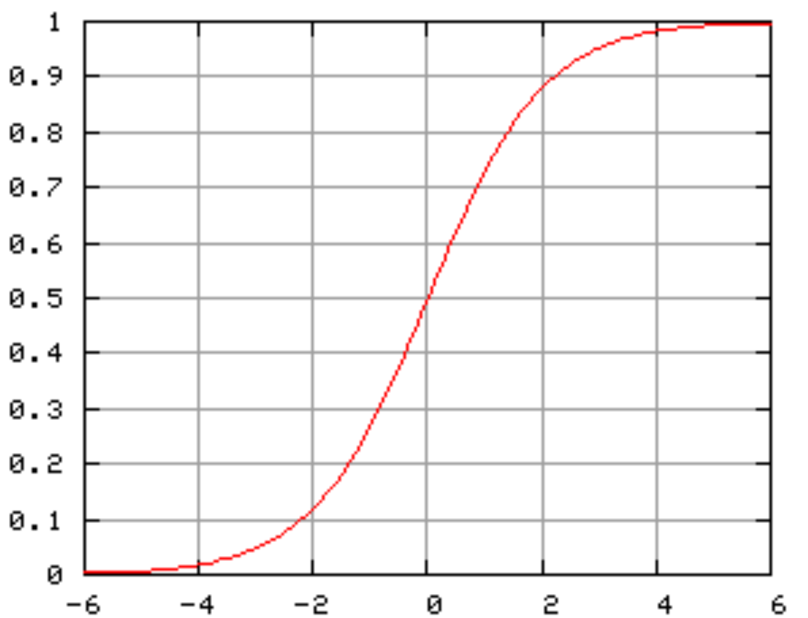
Logistic Regression

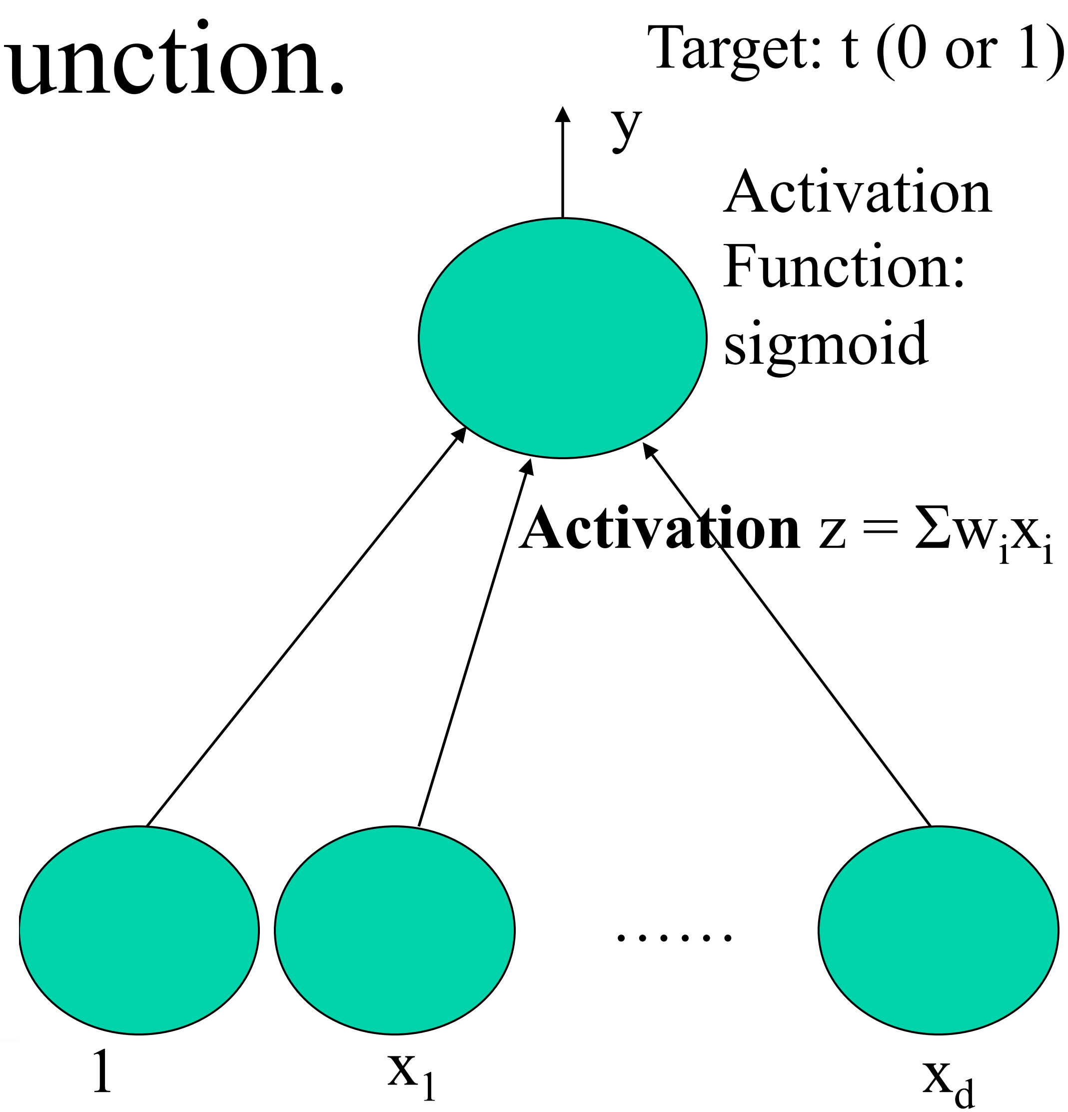
- Estimate posterior distribution: $P(C_1|x)$
- Dose – response estimation: in bioassay, the relation between dose level and death rate $P(\text{death} \mid x)$.
- We can not use 0/1 hard classification.
- We can not use unconstrained linear regression because $P(\text{death} \mid x)$ must be in $[0,1]$?

Logistic Regression and One Layer Neural Network With Sigmoid Function.

$$P(\text{death} \mid x) = \frac{1}{1 + e^{-wx}}$$

(Sigmoid function)





How to Adjust Weights?

- Minimize error E=(t-y)^2. For simplicity, we derive the formula for one data point. For multiple data points, just add the gradients together.

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial z}\frac{\partial z}{\partial w_i} = -2(t-y)y(1-y)x_i$$

Notice: $$\frac{\partial y}{\partial z} = \frac{\partial(\frac{1}{1+e^{-z}})}{\partial z} = \frac{1}{1+e^{-z}}(1-\frac{1}{1+e^{-z}}) = y(1-y)$$

Error Function and Learning

- Least Square
- Maximum likelihood: output y is the probability of being in C_1 (t=1). 1- y is the probability of being in C_2. So what is probability of $P(t|x) = y^t(1-y)^{1-t}$.
- Maximum likelihood is equivalent to minimize negative log likelihood:

  $E = -\log P(t|x) = -t\log y - (1-t)\log(1-y)$. (cross entropy)

How to Adjust Weights?

- Minimize error E= -tlog*y* - (1-*t*)log(1-*y*). For simplicity, we derive the formula for one data point. For multiple data points, just add the gradients together.

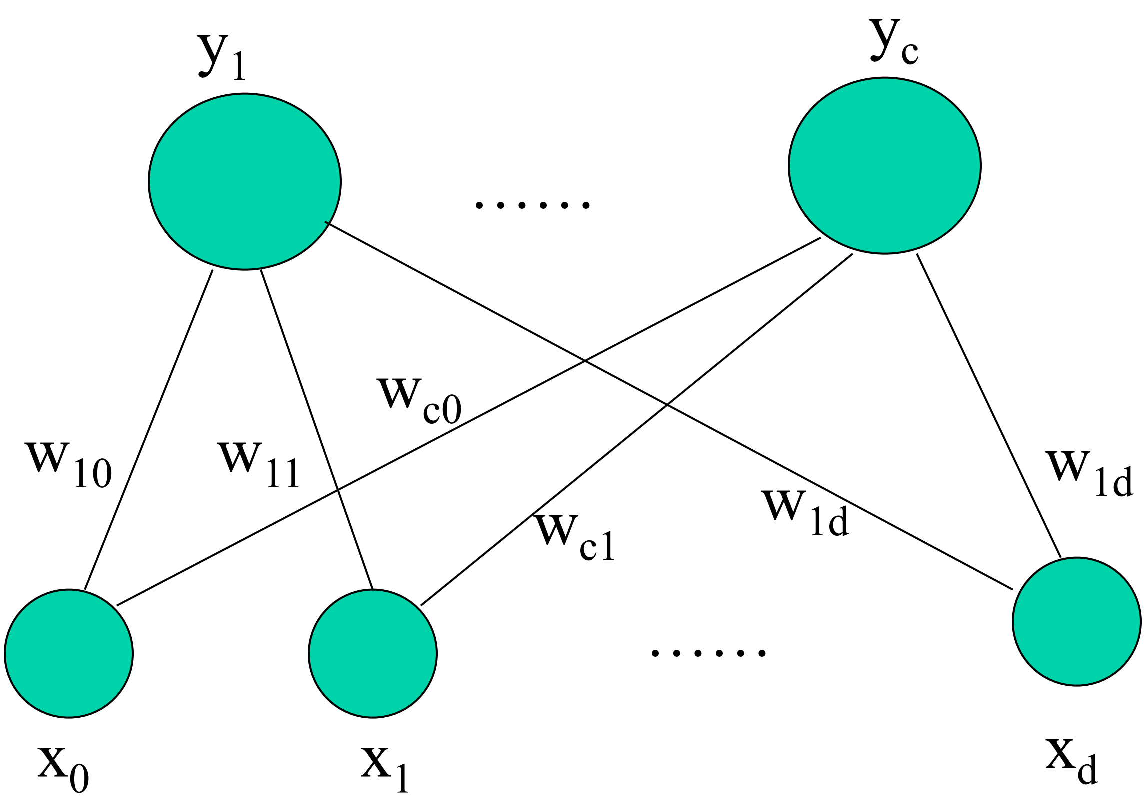
$$\frac{\partial E}{\partial y} = -\frac{t}{y} - \frac{1-t}{1-y}(-1) = -\frac{t}{y} - \frac{t-1}{1-y} = \frac{y-t}{y(1-y)}$$

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial z}\frac{\partial z}{\partial w_i} = \frac{y-t}{y(1-y)}y(1-y)x_i = (y-t)x_i$$

Update rule: $w_i^{(t+1)} = w_i^t + \eta(t-y)x_i$

Multi-Class Logistic Regression

- Transfer (or activation) function is normalized exponentials (or soft max)



$$y_i = \frac{e^{a_i}}{\sum_{j=1}^{c} e^{a_j}}$$

Activation Function

$$a_i = \sum_{j=0}^{d} w_{ij} x_j$$

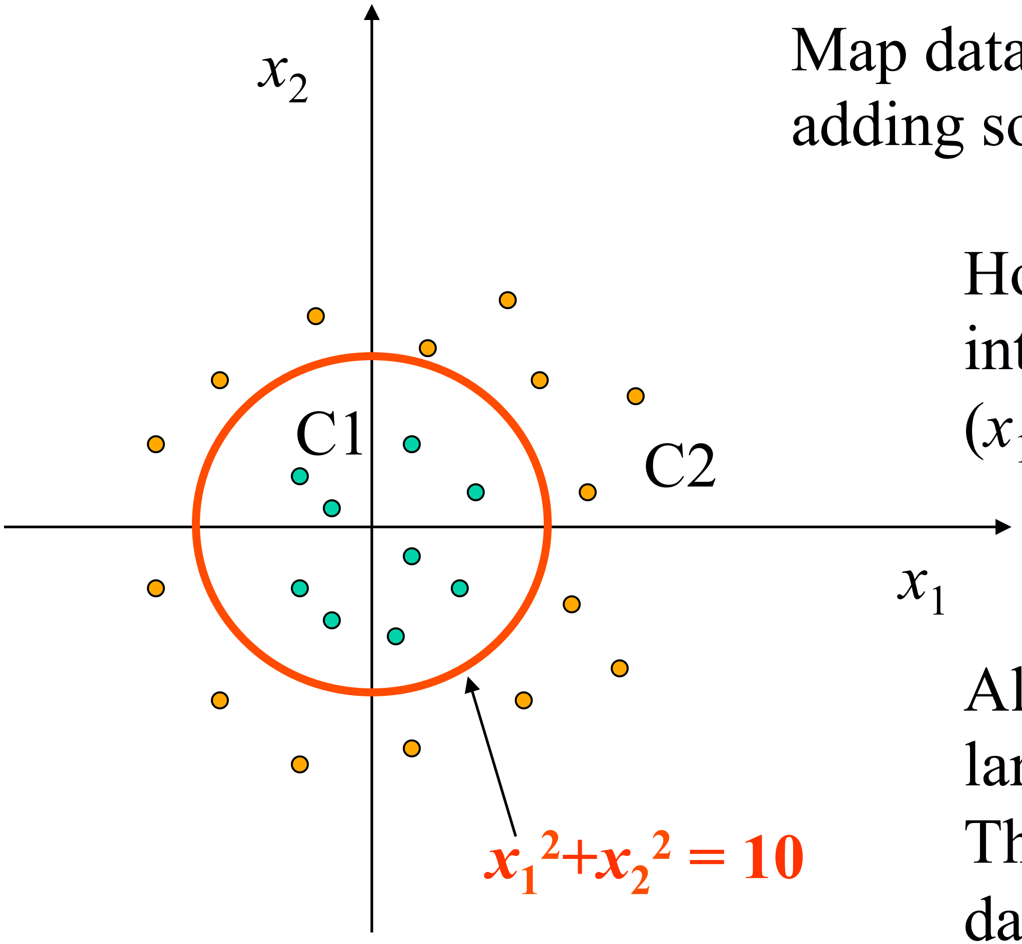
Activation to Node O_i

How to learn this network? Once again, gradient descent.

Questions?

- Is logistic regression a linear regression?
- Can logistic regression handle non-linearly separable data?
- How to introduce non-linearity?

Support Vector Machine Approach



Map data point into high dimension, e.g. adding some non-linear features.

How about we augument feature into three dimension (x_1, x_2, $x_1^2+x_2^2$).

All data points in class C2 have a larger value for the third feature Than data points in C1. Now data is linearly separable.

Neural Network Approach

- Multi-Layer Perceptrons
- In addition to input nodes and output nodes, some hidden nodes between input / output nodes are introduced.
- Use hidden units to learn internal features to represent data. Hidden nodes can learn internal representation of data that are not explicit in the input features.
- Transfer function of hidden units are non-linear function

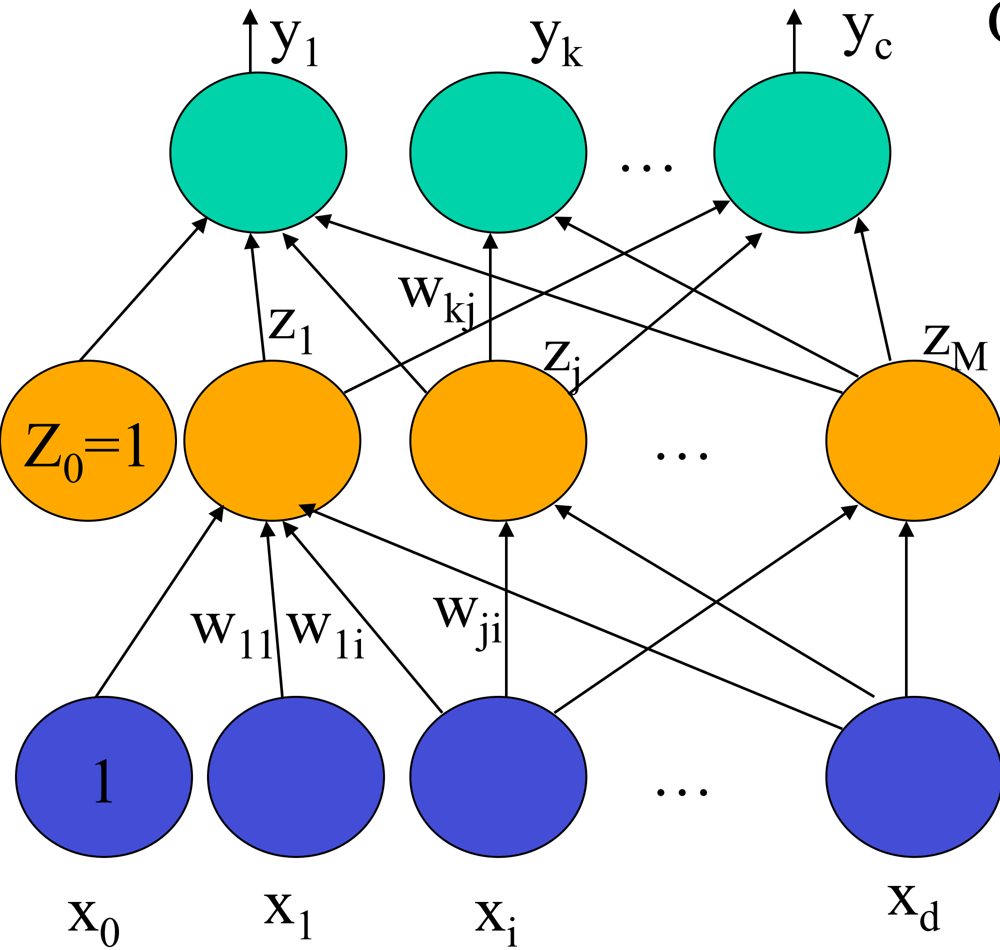
Multi-Layer Perceptron

- Connections go from lower layer to higher layer. (usually from input layer to hidden layer, to output layer)
- Connection between input/hidden nodes, input/output nodes, hidden/hidden nodes, hidden/output nodes are arbitrary as long as there is no loop (must be feed-forward).
- However, for simplicity, we usually only allow connection from input nodes to hidden nodes and from hidden nodes to output nodes. The connections with a layer are disallowed.

Multi-Layer Perceptron

- Two-layer neural network (one hidden and one output) with non-linear activation function is a **universal function approximator** (see Baldi and Brunak 2001 or Bishop 96 for the proof), i.e. it can approximate any numeric function with arbitrary precision given a set of appropriate weights and hidden units.
- Thus, we usually use two-layer (or three-layer if you count the input as one layer) neural network. Increasing the number of layers is occasionally helpful.

Two-Layer Neural Network



Output

Activation function: f (linear,sigmoid, softmax)

Activation of unit a_k: $\sum_{j=0}^{M} w_{kj} z_j$

Activation function: g (linear, tanh, sigmoid)

Activation of unit a_j: $\sum_{i=0}^{d} w_{ji} x_i$

$$y_k = f\left(\sum_{j=0}^{M} w_{kj} \times g\left(\sum_{i=0}^{d} w_{ji} x_i\right)\right)$$

Adjust Weights by Training

- How to adjust weights?
- Adjust weights using known examples (training data) (x_1,x_2,x_3,…,x_d,t).
- Try to adjust weights so that the difference between the output of the neural network y and t (target) becomes smaller and smaller.
- Goal is to minimize Error (difference) as we did for one layer neural network

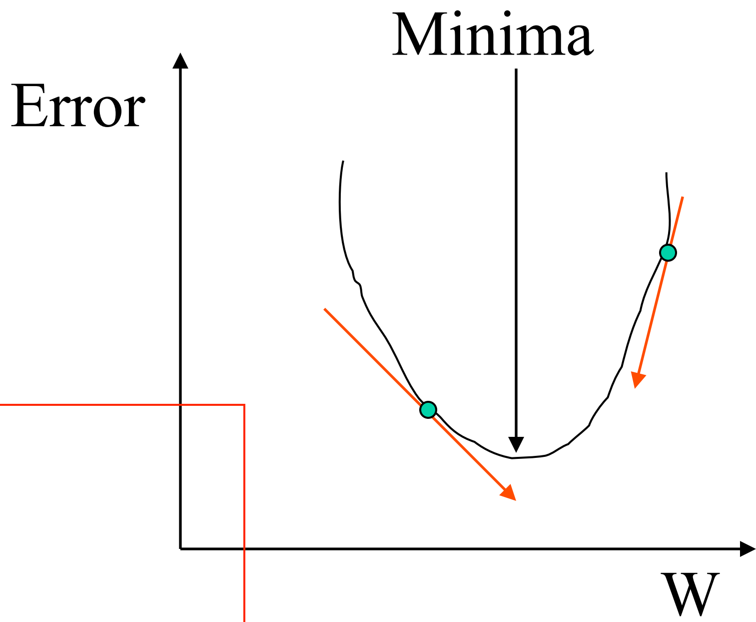
Adjust Weights using Gradient Descent (Back-Propagation)

Known:

Data: $(x_1,x_2,x_3,\ldots,x_n)$ target t.

Unknown weights w:

$w_{11}, w_{12}, \ldots$



Randomly initialize weights

Repeat

for each example, compute output y

calculate error $E = (y-t)^2$

compute the derivative of E over w: $dw = \frac{\partial E}{\partial w}$

$w_{\text{new}} = w_{\text{prev}} - \eta * dw$

Until error doesn't decrease or max num of iterations

Note: η is learning rate or step size.

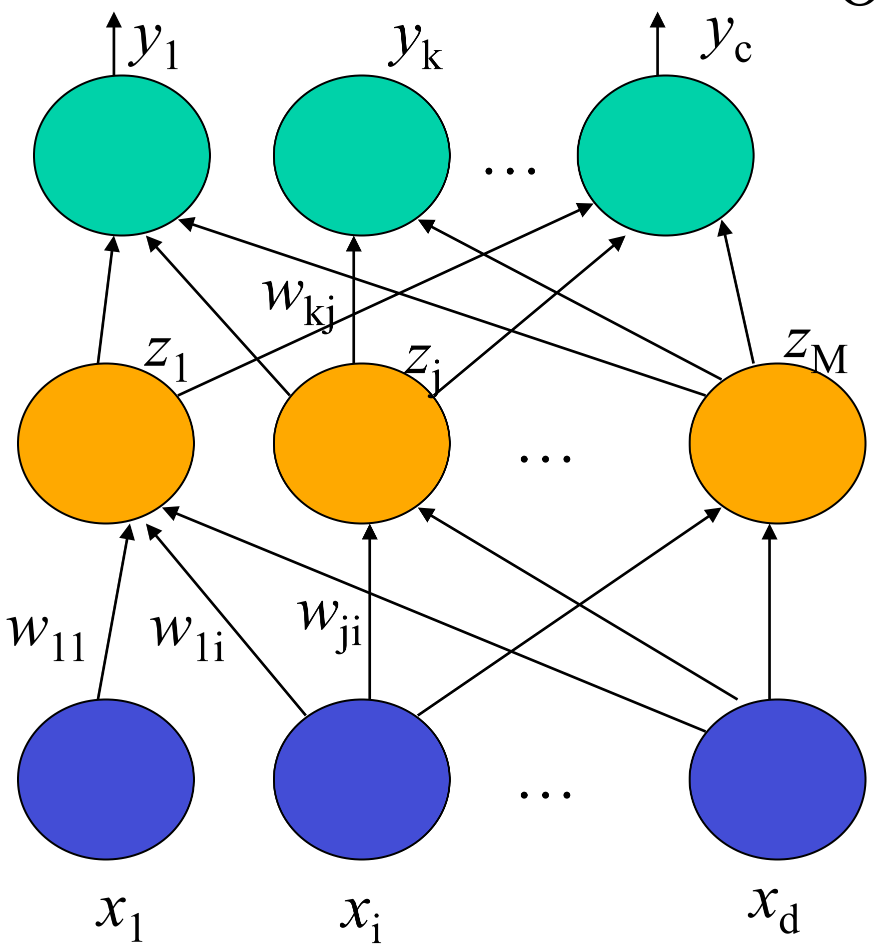
Insights

- We know how to compute the derivative of one layer neural network? How to change weights between input layer and hidden layer?
- Should we compute the derivative of each w separately or we can reuse intermediate results? We will have an efficient back-propagation algorithm.
- We will derive learning for one data example. For multiple examples, we can simply add the derivatives from them for a weight parameter together.

Neural Network Learning: Two Processes

- Forward propagation: present an example (data) into neural network. Compute activation into units and output from units.
- Backward propagation: propagate error back from output layer to the input layer and compute derivatives (or gradients).

Forward Propagation



Output y_k

Activation function: f (linear,sigmoid, softmax)

Activation of unit a_k: $\sum_{j=1}^{M} w_{kj} z_j$

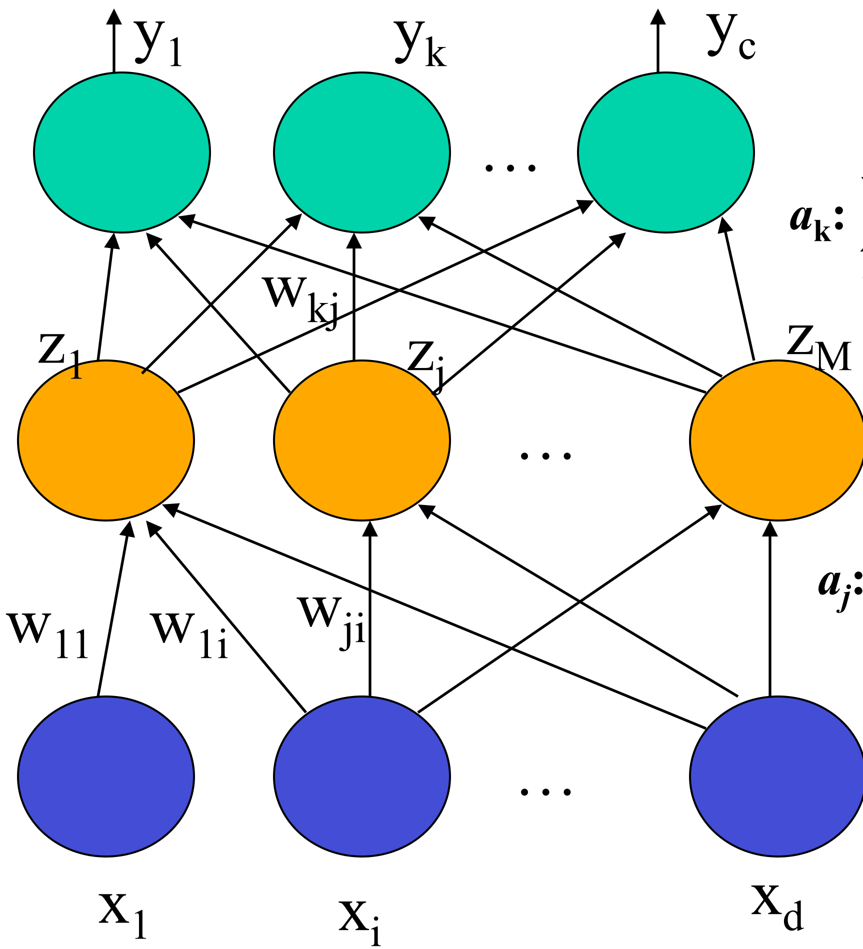
z_j

Activation function: g (linear, tanh, sigmoid)

Activation of unit a_j: $\sum_{i=1}^{d} w_{ji} x_i$

Time complexity?
O(dM + MC) = O(W)

Backward Propagation



f

a_k: $\sum_{j=1}^{M} w_{kj} z_j$

g

a_j: $\sum_{i=1}^{d} w_{ji} x_i$

$$E = \frac{1}{2}\sum_{k=1}^{C}(y_k - t_k)^2$$

$$\frac{\partial E}{\partial y_k} = y_k - t_k$$

$$\frac{\partial E}{\partial a_k} = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial a_k} = (y_k - t_k) f'(a_k) = \delta_k$$

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial a_k}\frac{\partial a_k}{\partial w_{kj}} = \delta_k z_j$$

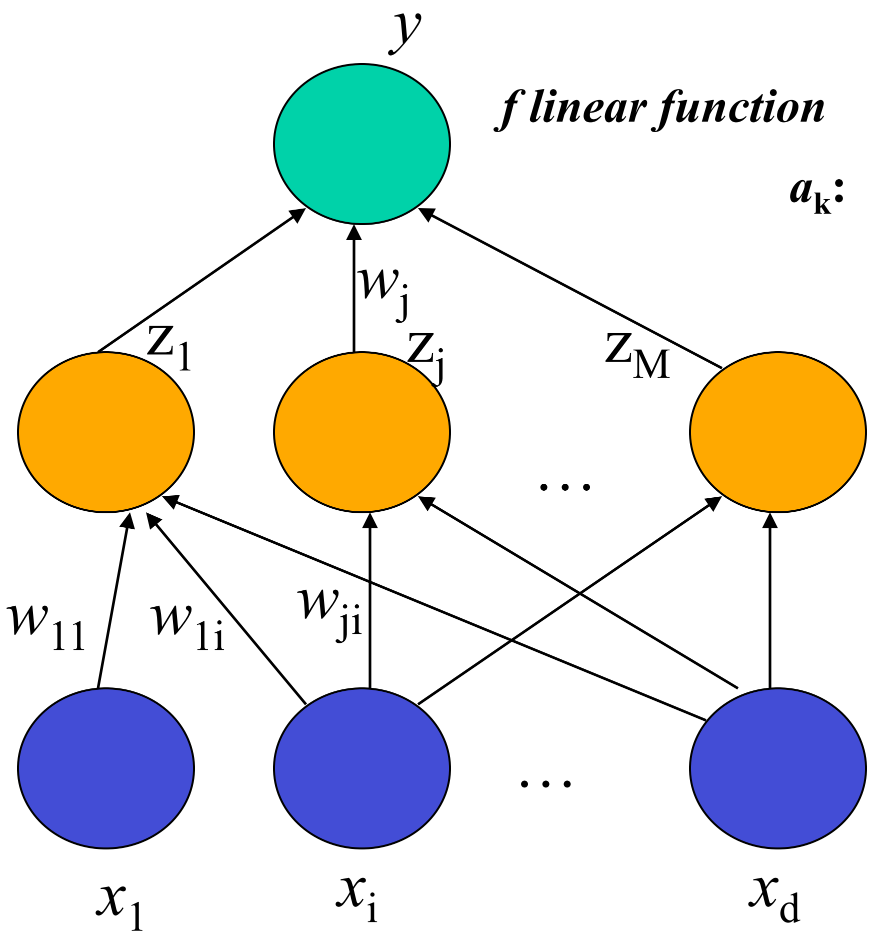
$$\frac{\partial E}{\partial a_j} = \sum_{k=1}^{c}\frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial a_k}\frac{\partial a_k}{\partial z_j}\frac{\partial z_j}{\partial a_j} = \sum_{k=1}^{C}\delta_k w_{kj} g'(a_j) = \delta_j$$

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial a_j}\frac{\partial a_j}{\partial wji} = \delta_j x_i$$

If no back-propagation, time complexity is: (MdC+CM)

Time complexity?
O(CM+Md) = O(W)

Example



f linear function

a_k:

g is sigmoid:

a_j: $\sum_{i=1}^{M} w_{ji} x_i$

$$E = \frac{1}{2}(y - t)^2$$

$$\delta = \frac{\partial E}{\partial a_k} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial a_k} = (y - t)$$

$$\frac{\partial E}{\partial w_j} = \delta z_j$$

$$\delta_j = \delta w_j g'(a_j) = (y - t) w_j z_j (1 - z_j)$$

$$\frac{\partial E}{\partial w_{ji}} = \delta_j x_i = (y - t) w_j z_j (1 - z_j) x_i$$

Algorithm

- **Initialize** weights w
- **Repeat**

  For each data point x, do the following:

  Forward propagation: compute outputs and activations

  Backward propagation: compute errors for each output units and hidden units. Compute gradient for each weight.

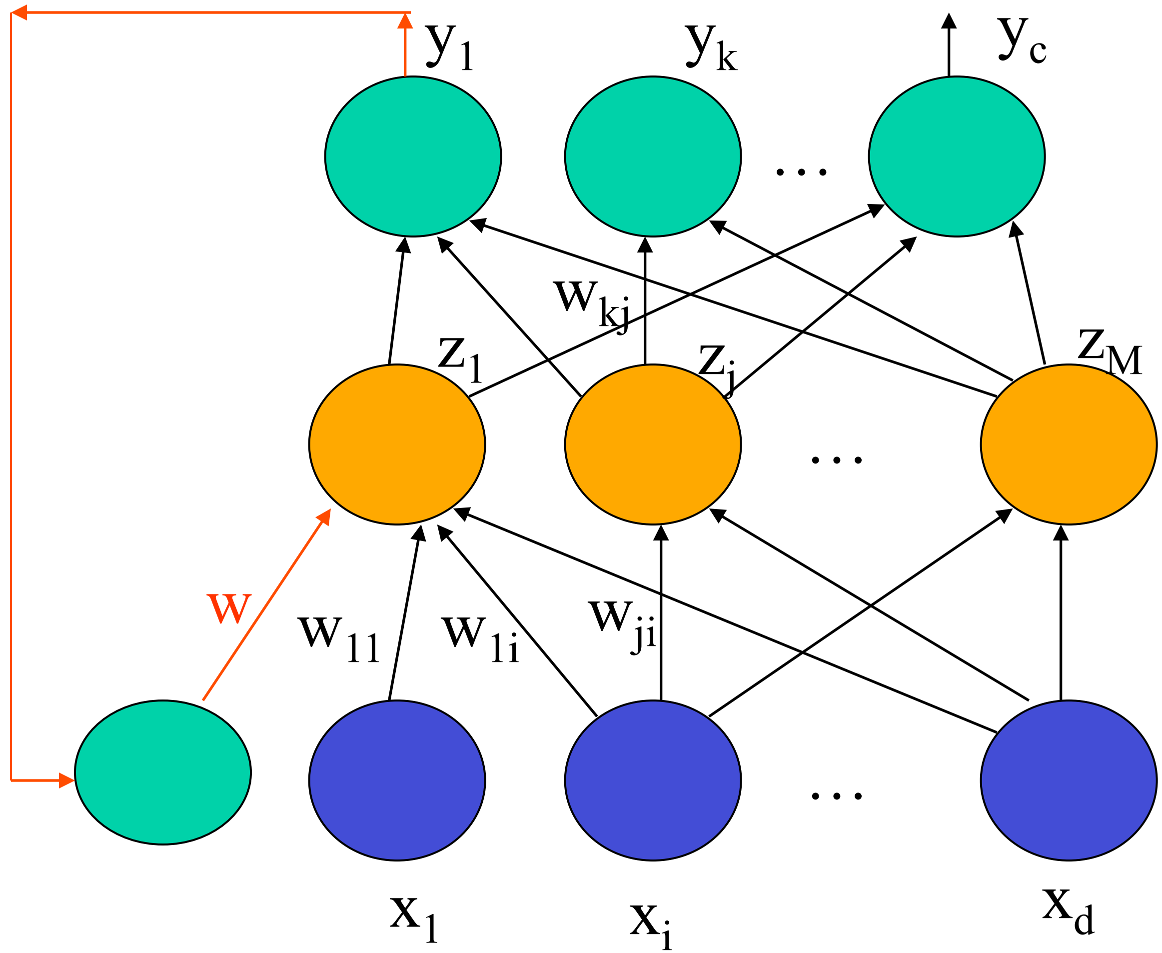
  Update weight $w = w - \eta \, (\partial E / \partial w)$

- **Until** a number of iterations or errors drops below a threshold.

Implementation Issue

- What should we store?
- An input vector x of *d* dimensions
- A M*d matrix $\{w_{ji}\}$ for weights between input and hidden units
- An activation vector of M dimensions for hidden units
- An output vector of M dimensions for hidden units
- A C*M matrix $\{w_{kj}\}$ for weights between hidden and output units
- An activation vector of C dimensions for output units
- An output vector of C dimensions for output units
- An error vector of C dimensions for output units
- An error vector of M dimensions for hidden units

Recurrent Network



Forward:
At time 1: present X1, 0
At time 2: present X2, y1
……

Backward:
Time t: back-propagate
Time t-1: back-propagate with
Output errors and errors from previous step

Recurrent Neural Network

1. Recurrent network is essentially a series of feed-forward neural networks sharing the same weights

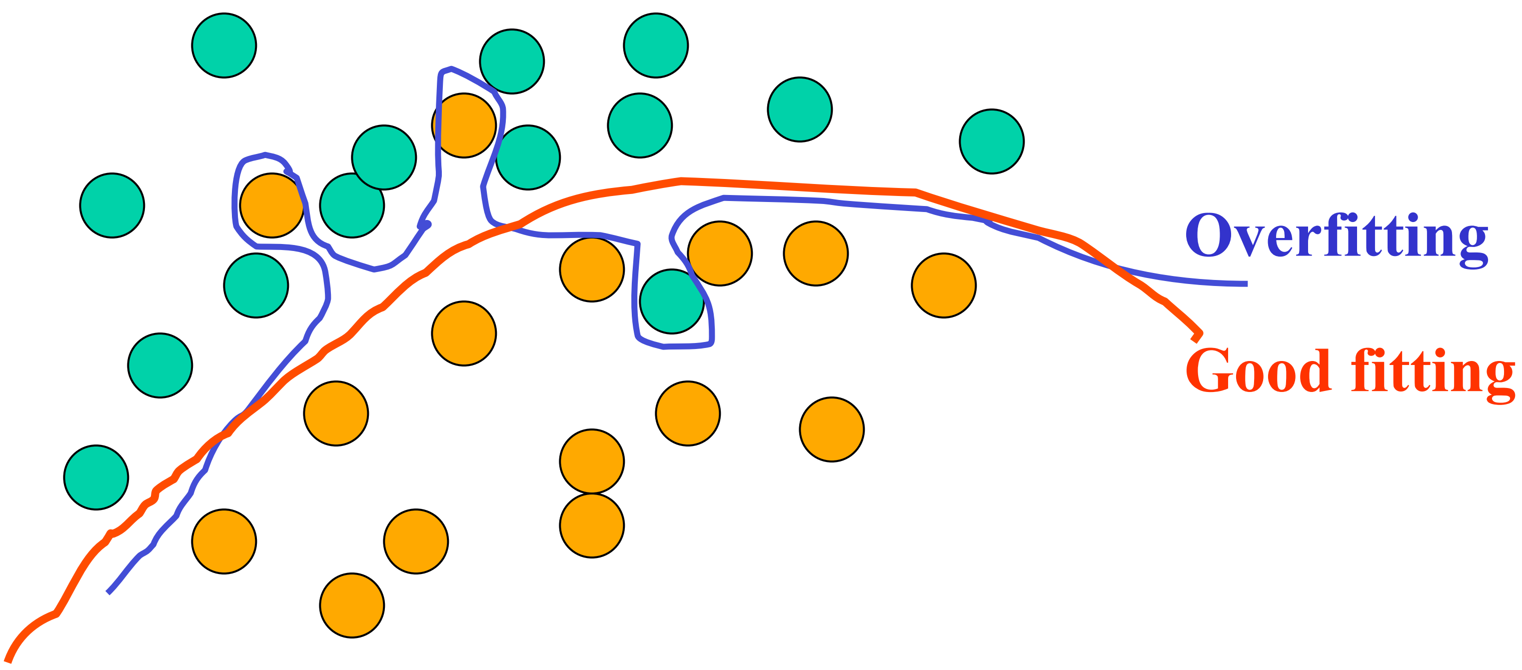
2. Recurrent network is good for time series data and sequence data such as biological sequences.

Overfitting

- The training data contains information about the regularities in the mapping from input to output. But it also contains noise
  - The target values may be unreliable.
  - There is sampling error. There will be accidental regularities just because of the particular training cases that were chosen.
- When we fit the model, it cannot tell which regularities are real and which are caused by sampling error.
  - So it fits both kinds of regularity.
  - If the model is very flexible it can model the sampling error really well. This is a disaster.

G. Hinton, 2006

Example of Overfitting and Good Fitting



Overfitting function can not generalize well to unseen data.

Preventing Overfitting

- Use a model that has the right capacity:
  - enough to model the true regularities
  - not enough to also model the spurious regularities (assuming they are weaker).
- Standard ways to limit the capacity of a neural net:
  - Limit the number of hidden units.
  - Limit the size of the weights.
  - Stop the learning before it has time to overfit.

G. Hinton, 2006

Limiting the Size of the Weights

- Weight-decay involves adding an extra term to the cost function that penalizes the squared weights.
  - Keeps weights small unless they have big error

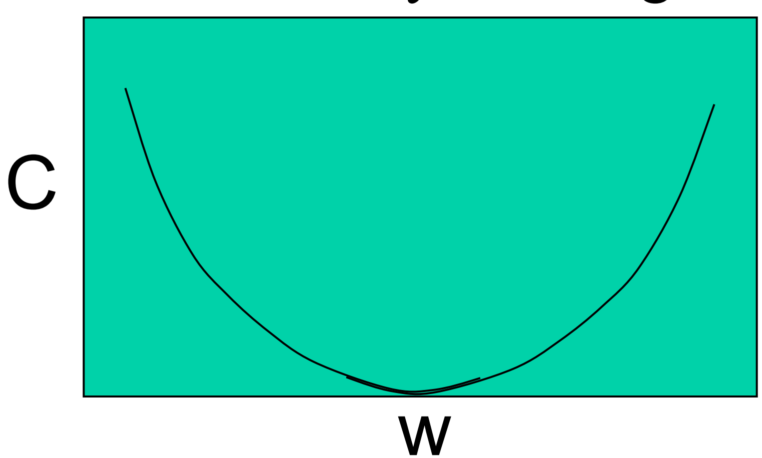
$$C = E + \frac{\lambda}{2}\sum_i w_i^2$$

The image cannot be displayed. Your computer may not have enough memory to open the image, or the image may have been corrupted. Restart your computer, and then open the file again. If the red x still appears, you may have to delete the image and then insert it again.

$$\frac{\partial C}{\partial w_i} = \frac{\partial E}{\partial w_i} + \lambda w_i$$

The image cannot be displayed. Your computer may not have enough memory to open the image, or the image may have been corrupted. Restart your computer, and then open the file again. If the red x still appears, you may have to delete the image and then insert it again.

$$when \quad \frac{\partial C}{\partial w_i} = 0, \quad w_i = -\frac{1}{\lambda}\frac{\partial E}{\partial w_i}$$



G. Hinton, 2006

The Effect of Weight-Decay

- It prevents the network from using weights that it does not need.
  - This can often improve generalization a lot.
  - It helps to stop it from fitting the sampling error.
  - It makes a smoother model in which the output changes more slowly as the input changes.
- If the network has two very similar inputs it prefers to put half the weight on each rather than all the weight on one.

G. Hinton, 2006

Deciding How Much to Restrict the Capacity

- How do we decide which limit to use and how strong to make the limit?
  - If we use the test data we get an unfair prediction of the error rate we would get on new test data.
  - Suppose we compared a set of models that gave random results, the best one on a particular dataset would do better than chance. But it wont do better than chance on another test set.
- So use a separate validation set to do model selection.

G. Hinton, 2006

Using a Validation Set

- Divide the total dataset into three subsets:
  - Training data is used for learning the parameters of the model.
  - Validation data is not used of learning but is used for deciding what type of model and what amount of regularization works best.
  - Test data is used to get a final, unbiased estimate of how well the network works. We expect this estimate to be worse than on the validation data.
- We could then re-divide the total dataset to get another unbiased estimate of the true error rate.

G. Hinton, 2006

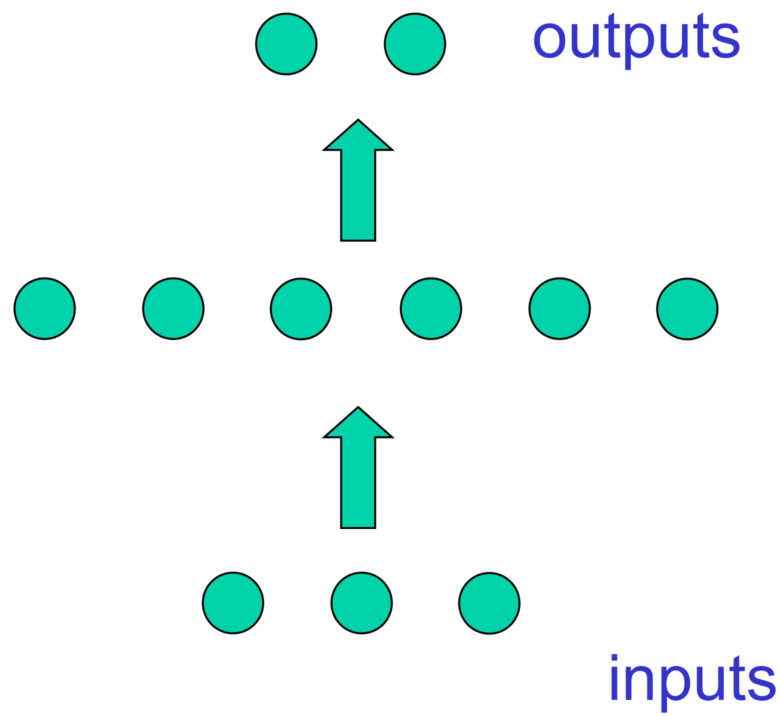
Preventing Overfitting by Early Stopping

- If we have lots of data and a big model, its very expensive to keep re-training it with different amounts of weight decay.
- It is much cheaper to start with very small weights and let them grow until the performance on the validation set starts getting worse (but don't get fooled by noise!)
- The capacity of the model is limited because the weights have not had time to grow big.

G. Hinton, 2006

Why Early Stopping Works

- When the weights are very small, every hidden unit is in its linear range.
  - So a net with a large layer of hidden units is linear.
  - It has no more capacity than a linear net in which the inputs are directly connected to the outputs!
- As the weights grow, the hidden units start using their non-linear ranges so the capacity grows.



G. Hinton, 2006

Combining Networks

- When the amount of training data is limited, we need to avoid overfitting.
  - Averaging the predictions of many different networks is a good way to do this.
  - It works best if the networks are as different as possible.
  - Combining networks reduces variance
- If the data is really a mixture of several different “regimes” it is helpful to identify these regimes and use a separate, simple model for each regime.
  - We want to use the desired outputs to help cluster cases into regimes. Just clustering the inputs is not as efficient.

How the Combined Predictor Compares with the Individual Predictors

- On any one test case, some individual predictors will be better than the combined predictor.
  - But different individuals will be better on different cases.
- If the individual predictors disagree a lot, the combined predictor is typically better than all of the individual predictors when we average over test cases.
  - So how do we make the individual predictors disagree? (without making them much worse individually).

G. Hinton, 2006

Ways to Make Predictors Differ

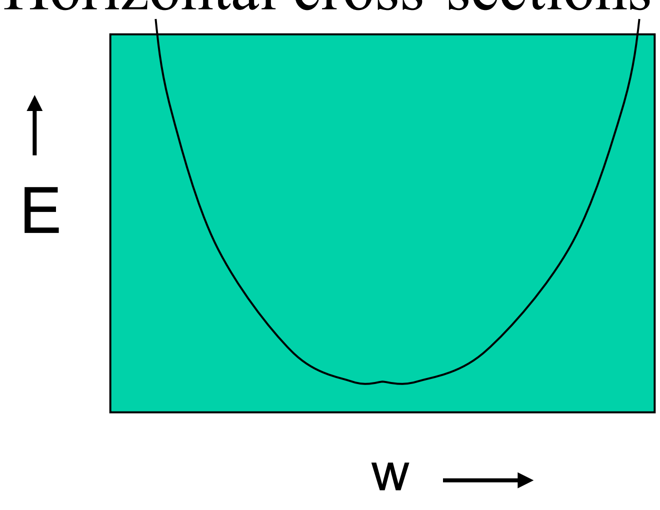
- Rely on the learning algorithm getting stuck in a different local optimum on each run.
  - A dubious hack unworthy of a true computer scientist (but definitely worth a try).
- Use lots of different kinds of models:
  - Different architectures
  - Different learning algorithms.
- Use different training data for each model:
  - Bagging: Resample (with replacement) from the training set: a,b,c,d,e -> a c c d d
  - Boosting: Fit models one at a time. Re-weight each training case by how badly it is predicted by the models already fitted.
    - This makes efficient use of computer time because it does not bother to "back-fit" models that were fitted earlier.

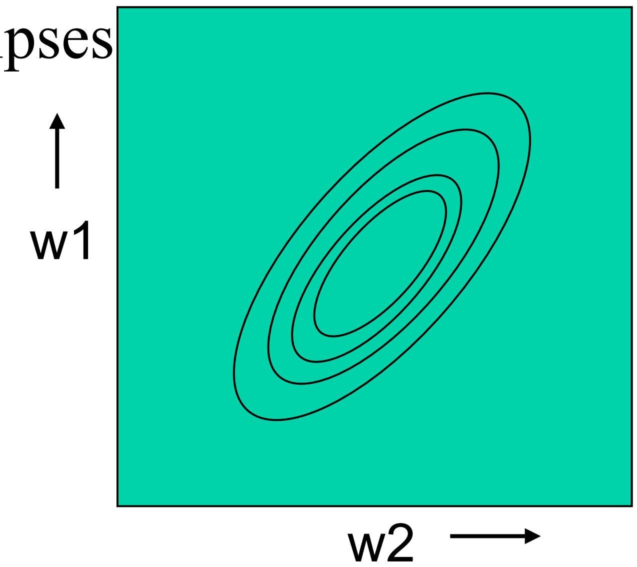
G. Hinton, 2006

How to Speedup Learning?
The Error Surface for a Linear Neuron

- The error surface lies in a space with a horizontal axis for each weight and one vertical axis for the error.
  - It is a quadratic bowl.
    - i.e. the height can be expressed as a function of the weights without using powers higher than 2. Quadratics have constant curvature (because the second derivative must be a constant)
  - Vertical cross-sections are parabolas.
  - Horizontal cross-sections are ellipses

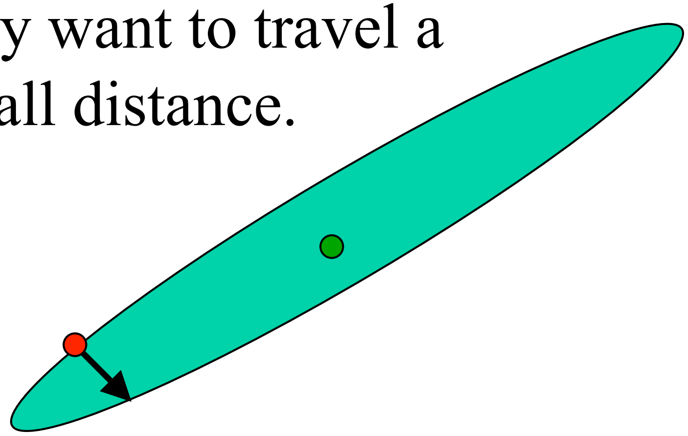
G. Hinton, 2006





Convergence Speed

- The direction of steepest descent does not point at the minimum unless the ellipse is a circle.
  - The gradient is big in the direction in which we only want to travel a small distance.
  - The gradient is small in the direction in which we want to travel a large distance.



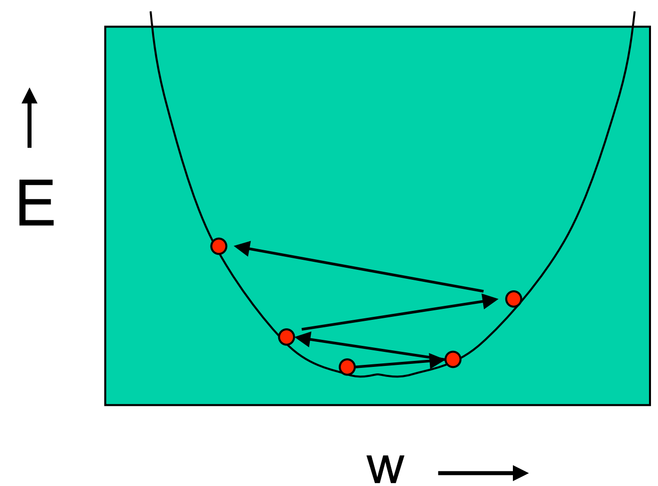
$$\Delta w_i = -\varepsilon \frac{\partial E}{\partial w_i}$$

This equation is sick. The RHS needs to be multiplied by a term of dimension w^2 to make the dimensions balance.

G. Hinton, 2006

How the Learning Goes Wrong

- If the learning rate is big, it sloshes to and fro across the ravine. If the rate is too big, this oscillation diverges.
- How can we move quickly in directions with small gradients without getting divergent oscillations in directions with big gradients?



G. Hinton, 2006

Five Ways to Speed up Learning

- Use an adaptive global learning rate
  - Increase the rate slowly if its not diverging
  - Decrease the rate quickly if it starts diverging
- Use separate adaptive learning rate on each connection
  - Adjust using consistency of gradient on that weight axis
- Use momentum
  - Instead of using the gradient to change the position of the weight "particle", use it to change the velocity.
- Use a stochastic estimate of the gradient from a few cases
  - This works very well on large, redundant datasets.
- Don't go in the direction of steepest descent.
  - The gradient does not point at the minimum.
    - Can we preprocess the data or do something to the gradient so that we move directly towards the minimum?

G. Hinton, 2006

The Momentum Method

Imagine a ball on the error surface with velocity v.

- It starts off by following the gradient, but once it has velocity, it no longer does steepest descent.
  - It damps oscillations by combining gradients with opposite signs.
  - It builds up speed in directions with a gentle but consistent gradient.
  - On an inclined plane it reaches a terminal velocity.

$$v(t) = \alpha\, v(t-1) - \varepsilon \frac{\partial E}{\partial w}(t)$$

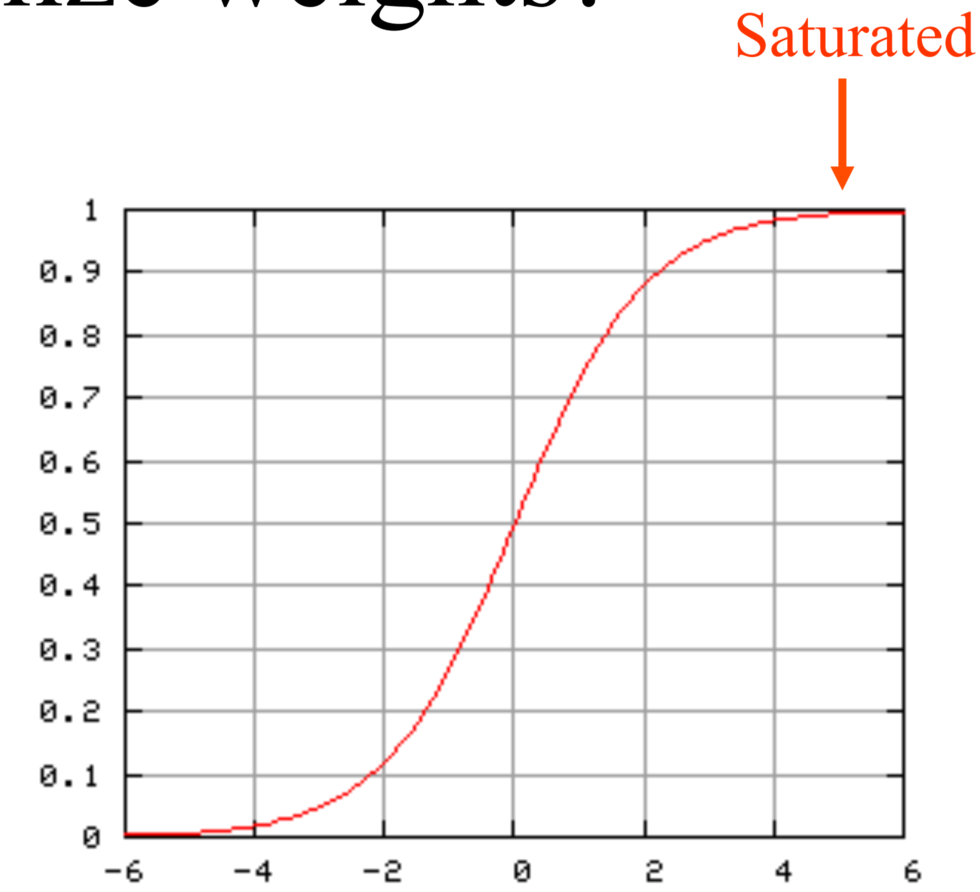
$$\begin{aligned} \Delta w(t) &= v(t) \\ &= \alpha\, v(t-1) - \varepsilon \frac{\partial E}{\partial w}(t) \\ &= \alpha\, \Delta w(t-1) - \varepsilon \frac{\partial E}{\partial w}(t) \end{aligned}$$

$$v(\infty) = \frac{1}{1-\alpha}\left(-\varepsilon \frac{\partial E}{\partial w}\right)$$

G. Hinton, 2006

How to Initialize weights?

- Use small random numbers. For instance small numbers between [-0.2, 0.2].
- Some numbers are positive and some are negative.
- Why are the initial weights should be small?



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

Neural Network Software

- **Weka** (Java): http://www.cs.waikato.ac.nz/ml/weka/

- **NNClass** and **NNRank** (C++): http://www.eecs.ucf.edu/~jcheng/cheng_software.html

  J. Cheng, Z. Wang, G. Pollastri. A Neural Network Approach to Ordinal Regression. IJCNN, 2008

NNClass Demo

- Abalone data:
  http://archive.ics.uci.edu/ml/datasets/Abalone

UCI Machine Learning Repository

Abalone (from Spanish *Abulón*) are a group of shellfish (mollusks) in the family **Haliotidae** and the ***Haliotis*** genus. They are marine snails