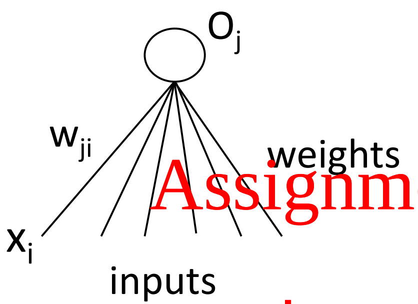
L19 --- Neural Nets I

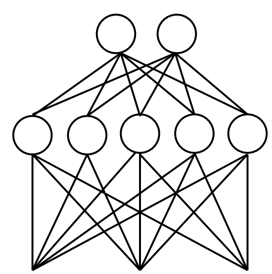
What's a Neural Network?

- Exceedingly simple processing units.



$$O_j = \sigma\left(w_{j0} + w_{j1}\, x_{j1} + w_{j2}\, x_2 + \ldots + w_{jN}\, x_N\right)$$
$$= \sigma\left(\sum_{i=0}^{N} w_{ji}\, x_i\right) = \sigma\left(\vec{w} \cdot \vec{x}\right), \qquad (x_0 = 1)$$

- Parallel colle ts provides a map from vector inputs to vecto



GEORGETOWN UNIVERSITY

Characteristics

- *Enormous flexibility* in maps that can be represented.
- *Mapping can be learned from examples*.
- Generalize far better than models that are linear in the parameters
- Relatively forgivi
- Extrapolate grac
- Training times c y hours for large nets with large datasets.
- Evaluation of learned function is *fast*
- Doesn't require programming in target function.
- Success depends on picking appropriate features for inputs x_i, and representation for output.

What Are They Good For?

- *Enormously flexible, can achieve huge range of maps*.
- *Mapping can be learned from examples.*

- Pattern Classification (statistical pattern recognition)
  - Text-to-speech
  - Handwritten, machine printed (OCR), cursive writing (online) recognition
  - Event dete eriments
  - Medical screening *Papnet*, adj tional screening, reduces false negatives

    http://www.mda. pdf/pap.pdf

    (testing on sputum smears too).
  - Acoustic front end for speech recognition systems.
  - Illegal drug source identification

What Are They Good For?

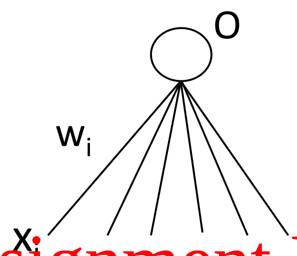
- Regression / prediction of continuous-valued systems
  - Time series prediction, e.g. financial forecasting
  - Non-linear regression
- Control
  - Plasma fusion reactor control
  - Chemical p
  - Quality con
  - Trailer truc
    http://www.handshake.de/user
  - Aircraft controller – recovery fr irframe
- Signal Processing
  - Adaptive noise cancellation
  - Adaptive vibration cancellation
  - Image analysis

GEORGETOWN UNIVERSITY

Why "Neural"?

- Artificial neural network (ANN) function is derived from massive connectivity, real nervous systems are also massively connected.
- Parallelism exploited - biological proces times on orde s. we make com spo in several tent dozen "processing steps".
- ANN units are cartoons of real neurons. The latter have complex dynamics, and can have tens of thousands of inputs (in cortex). Real nervous systems have a multitude of neuron types.

Adaptive Linear Unit (Adaline)



$$O = \vec{w} \cdot \vec{x} = \sum_{i=0}^{N} w_i \ x_i$$

- Training – a tches target values in lea se
  - Data : input / target pairs $\{\vec{x}_d, t_d\}$ $d = 1, \ldots, D$
  - Performance metric, or *cost function* – mean squared error $\mathcal{E}(\vec{w}) = \frac{1}{2D} \sum_{d=1}^{D} (t_d - O(\vec{x}_d))^2 = \frac{1}{2D} \sum_{d=1} (t_d - \vec{w}$

GEORGETOWN UNIVERSITY

Linear Unit – Gradient Descent

– Optimization: crawl downhill (steepest descent) on the error surface

$$\vec{w} \leftarrow \vec{w} + \Delta\vec{w} \quad or \quad w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = -\eta \frac{\partial \mathcal{E}(\vec{w})}{\partial w_i} \quad or \quad \Delta\vec{w} = -\eta \, \nabla \mathcal{E}(\vec{w})$$

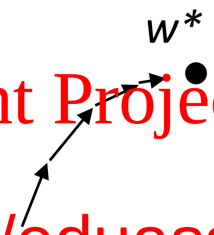
$$\frac{\partial \mathcal{E}(\vec{w})}{\partial w_i} = \frac{1}{D}\sum_{d=1}^{D}(t_d - \vec{w}\cdot\vec{x}_d)(-x_{id}) = \frac{1}{D}\sum_{d=1}^{D}(t_d - O(\vec{x}_d))\ (-x_{id})$$

So

$$\boxed{\Delta w_i = \eta \ \frac{1}{D}\sum_{d=1}^{D}(t_d - O(\vec{x}_d))\ x_{id}}$$

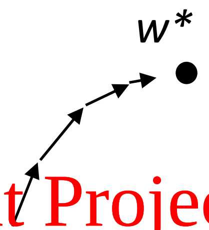
Linear Unit – Gradient Descent

- The error function E(*w*) is quadratic in *w,* and bounded below by zero. There is unique, global minimum *w*.*



- Can show that for learning rate η sufficiently small, this algorithm will approach *w** exponentially. How small? Must have

Linear Unit – Gradient Descent



- Can show that for learning rate η s ll, this algorithm will approach w^* exponentially. How small? Must have

$$0 < \eta < \frac{2}{\lambda}$$

where λ is the largest eigenvalue of the autocorrelation matrix

$$R = \frac{1}{D} \sum_{d=1}^{D} x_d \, x_d^T \qquad \text{i.e.} \qquad R_{ij} = \frac{1}{D} \sum_{d=1}^{D} x_{di} \, x_{dj}$$

GEORGETOWN UNIVERSITY

Linear Unit
Stochastic Gradient Descent

- Gradient descent

$$\Delta w_i = \eta \ \frac{1}{D} \sum_{d=1}^{D} (t_d - O(\vec{x}_d)) \ x_{id} \qquad \text{A}$$

- Instead of summing over all data pairs for each update to *w,* just use <u>one</u> data pair for each u n input/target pair $\{ \overline{x_d}, t_d \}$ at random from the odify *w*

$$\boxed{\Delta w_i = \eta \ (t_d - O(\vec{x}_d)) \ x_{id}}$$

This is the celebrated *Widrow-Huff* or Mean Square) algorithm.

- – Note that the gradient descent (A) is the *average* of this stochastic gradient descent (B), over all training data.
- – The stochastic descent is a *noisy* version of the true gradient descent.

Stochastic vs True Gradient Descent

true gradient descent

stochas
gradient descent

Linear Unit with LMS Training

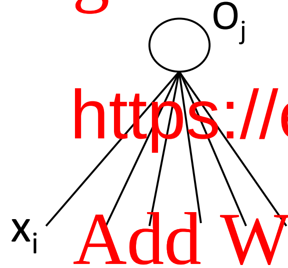
- Used in adaptive filter applications: adaptive noise cancellation and vibration damping, linear prediction problems (linear regression, AR models).

Perceptron Classifier

- Early ANN -- Rosenbaltt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*, Spartan, 1962.

  Single unit with <u>hard-limiter</u> output

$$O \qquad \left(Nin \qquad \right)$$



$$\sigma(y) = \begin{cases} +1 & y > 0 \\ -1 & y \le 0 \end{cases}$$

$$O(\vec{x}) = \mathrm{sgn}(\vec{w} \cdot \vec{x})$$

- Represents a *dichotomy* respon – to input vector. Input is member of class (+1) or not (-1). Concept is present (+1), or not (-1).

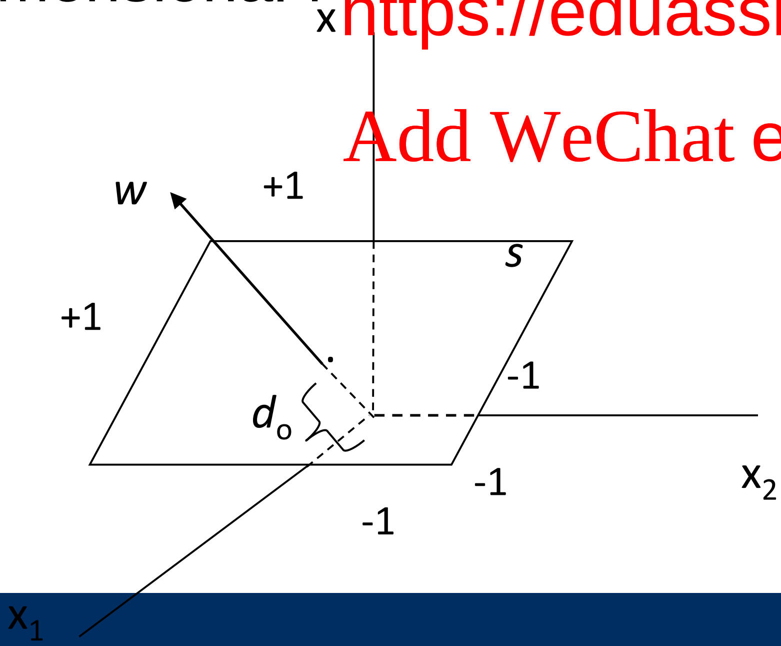
  e.g. – Does this picture have a tree in it? This is tough, the inputs *x* will need to be superbly crafted features.

Perceptron Classifier - Geometry

Hypothesis space is space of all possible weights *w* (R^{N+1})

Learning means choosing weight vector *w* that correctly classifies the training data.

Perceptron weight vector defines a *hyperplane* *s* in the N-dimensional f



$$w \cdot x = \sum_{i=0}^{N} w_i \, x_i$$

$$w \perp s$$

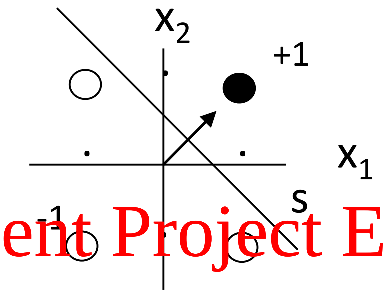
$$d_0 = \frac{-w_0}{|w|}$$

$$O(\vec{x}) = \mathrm{sgn}(\vec{w} \cdot \vec{x})$$

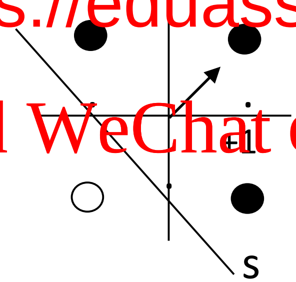
Perceptron Limitations

Boolean functions

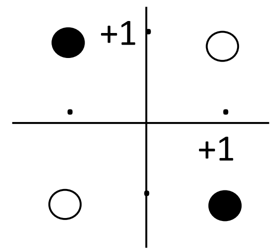
AND



OR



XOR



can only solve *linearly separable* dichotomies

GEORGETOWN UNIVERSITY

Perceptron Learning

- Training data input / target pairs (e.g. pictures with trees, +1 target, and pictures without trees, -1 target) $\{ x_d, t_d \}$
- We want $\vec{w} \cdot \vec{x}_d > 0 \quad for \quad t_d = +1$

$$\vec{w} \cdot \vec{x}_d < 0 \quad for \quad t_d = -1$$

this is equivale

$$(\vec{w} \cdot \vec{x}_d)\, t_d > 0 \quad \text{for all data}$$

A given data example will be mi f $(\vec{w} \cdot \vec{x}_d)\, t_d < 0$

- Define cost function $\mathcal{E}(\vec{w}) = \sum_{misclassified} -\left(\vec{w} \cdot \vec{x}_d\right) t_d \geq 0$
- Do stochastic gradient descent on this cost function : <u>If the example x_d is misclassified,</u> change the weights according to

$$\boxed{\Delta w_i = \eta \quad t_d \, x_{id}}$$

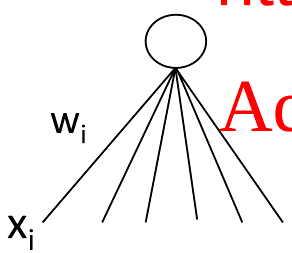
Perceptron Learning

- If the data are *linearly separable*, this algorithm will converge, in a finite number of steps, to a weight that correctly classifies *all* the training data.

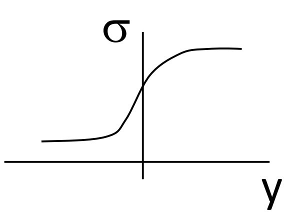
Soft Threshold
Differentiable "Perceptron"

- In order to get past the restriction to *linearly separable* problems, we are going to combine many *non-linear* neurons. (Why non-linear?)

- In order to tr rks, we introduce a *s*



$$O = \sigma(\vec{w}\cdot\vec{x}) = \sigma\left(\sum_{i=0}^{N} w_i x_i\right)$$



Smooth, bounded, monotonically increasing.

Sigmoidal Functions

- Typical choices are
  - Logistic function

$$\sigma(y) = \frac{1}{1+\exp(-y)}$$

  - Hyperbolic tangent

$$\sigma(y) = \tanh(y)$$

Training the Soft Threshold

Logistic function – targets are {0,1}
Hyperbolic tangent – targets are {-1,1}

Cost function $$\mathcal{E}(\vec{w}) = \frac{1}{2D} \sum_{d=1}^{D} (t_d - O(\vec{x}_d))^2 = \frac{1}{2D} \sum_{d=1}^{D} (t_d - \sigma(\vec{w} \cdot \vec{x}_d))^2$$

Train by gradient

$$\Delta w_i = -\eta \frac{\partial \mathcal{E}(\vec{w})}{\partial w_i}$$

$$\frac{\partial \mathcal{E}(\vec{w})}{\partial w_i} = \frac{1}{D} \sum_{d=1}^{D} (t_d - O(\vec{x}_d)) \frac{\partial O(\vec{x}_d)}{\partial w_i} = \frac{1}{D} \sum_{d=1}^{D} (t_d - O(\vec{x}_d)) \frac{\partial \sigma(\vec{w} \cdot \vec{x}_d)}{\partial w_i}$$

$$= \frac{1}{D} \sum_{d=1}^{D} (t_d - O(\vec{x}_d))\, \sigma'(\vec{w} \cdot \vec{x}_d) \frac{\partial \vec{w} \cdot \vec{x}_d}{\partial w_i} = \frac{1}{D} \sum_{d=1}^{D} (t_d - O(\vec{x}_d))\, \sigma'(\vec{w} \cdot \vec{x}_d)\, x_{di}$$

So

$$\boxed{\Delta w_i = \eta \frac{1}{D} \sum_{d=1}^{D} (t_d - O(\vec{x}_d))\, \sigma'(\vec{w} \cdot \vec{x}_d)\, x_{id}}$$

Training the Soft Threshold

- We have the gradient descent rule

$$\Delta w_i = \eta \ \frac{1}{D} \sum_{d=1}^{D} (t_d - O(\vec{x}_d)) \ \sigma'(\vec{w} \cdot \vec{x}_d) \ \ x_{id}$$

just like the linear gradient descent
except for slo

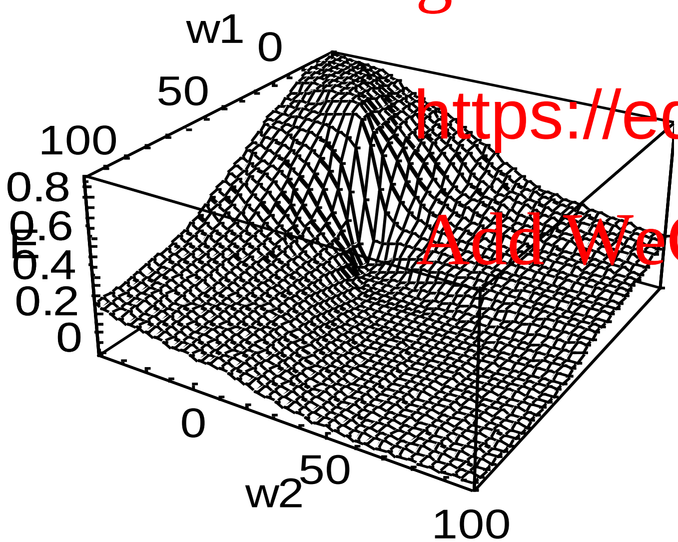
- Stochastic gra

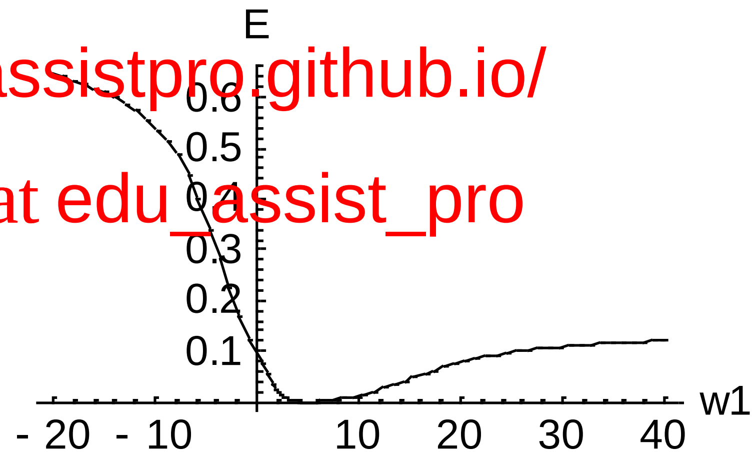
$$\Delta w_i = \eta \ (t_d - O(\vec{x}_d)) \ \sigma'(\vec{w} \cdot \vec{x}_d) \ \ x_{id}$$

- Note that if we get up onto the flat “rails” of the sigmoid, then the slope σ’ gets very small, and the gradient of the cost function gets very small → slow progress.

Cost Function

- The cost surface is now not a simple parabolic function, but instead is more complex looking.

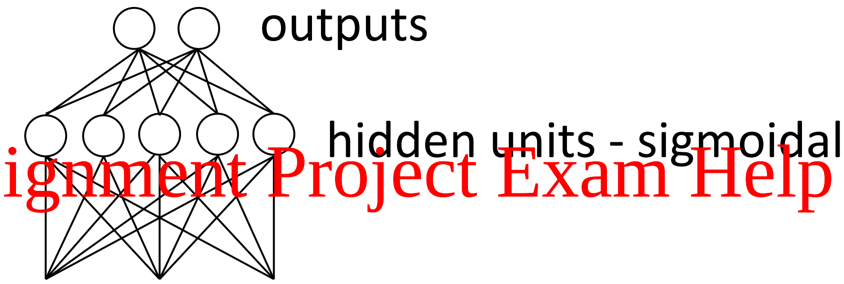




Workhorse Neural Networks
Multi-Layer Perceptrons (MLP)

Feed forward, layered networks, with sigmoidal hidden units

.



Can have more than two layers of weights

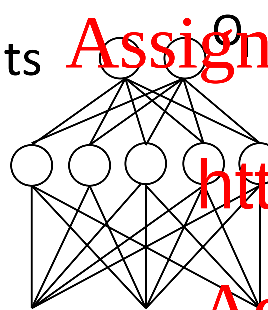
Output nodes

- Linear for regression, time series prediction, other problems needing full range of real values in output.
- Sigmoidal for classification problems.

Number of inputs, number of outputs determined by problem.
Number of hidden units is an <u>architectural parameter.</u>
More hidden nodes → more functions available.

MLP Output

- Signal propagation (forward pass, bottom-up)



$$O_l = g(\sum_k w_{lk}$$

$$with \quad g(net_l) = \begin{cases} \sigma(net_l) & for\ clas \\ net_l & for\ regr \end{cases}$$

$$y_k = \sigma\ (\sum_i w_{ki}\ x_i) \equiv \sigma(net_k)$$

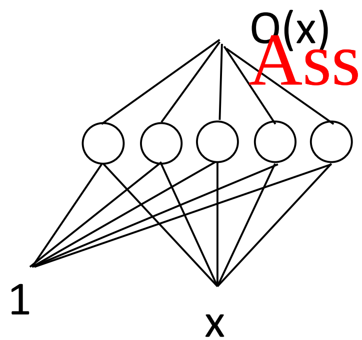
GEORGETOWN UNIVERSITY

Example Uses

- Classification – e.g. from text fig 4.5. Able to produce <u>non-linear</u> class boundaries.
- Fisher Iris data:

Example Uses

- Non-linear regression



able to map strongly
non-linear functions

Gradient Descent in MLP

- Cost function as before:

number of outputs

$$\boxed{\mathcal{E}(\vec{w}) = \frac{1}{2D}\sum_{d=1}^{D}\sum_{m=1}^{N_O}\left(t_{dm} - O_m(\vec{x}_d)\right)^2}$$

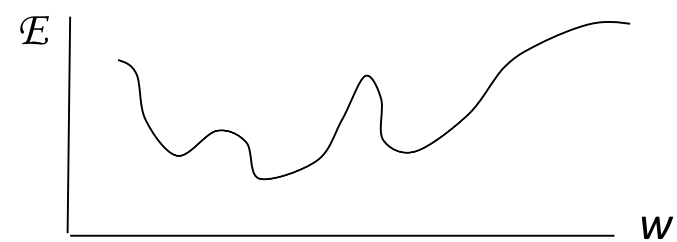
- Learning by gradient descent

$$\Delta w_{ij} = -\eta \frac{\partial \mathcal{E}(\vec{w})}{\partial w_{ij}}$$

- Calculating gr

- Surface can have multiple loc Some may have lower cost than others.

- Local optima are in different basins of attraction; where you end depends on where you start.



Stochastic Gradient Descent in MLP

As above, but <u>no</u> sum over data pairs *d*

$$\boxed{\mathcal{E}_d(\vec{w}) = \frac{1}{2}\sum_{m=1}^{N_O}\left(t_{dm} - O_m(\vec{x}_d)\right)^2}$$

$$\Delta w_{ij} = -\eta \frac{\partial \mathcal{E}_d(\vec{w})}{\partial w_{ij}}$$

Stochastic descent has some robustne etting stuck in poor local minima. Where you end, depends on where you start, learning rate, <u>and</u> the order the examples are given.

Can also be faster in clock-time for large data sets. Instead of waiting to accumulate errors from all data before making a weight change, make a small weight change in response to each datum.

Visualization of Stochastic Gradient Descent

- Different 2-d slices through *E(w)* for 9-d weight space
  - eg 1 Assignment Project Exam Help

https://eduassistpro.github.io/

Add WeChat edu_assist_pro

Visualization of Stochastic Gradient Descent

– eg 2

Next

- Backpropagation training of MLP.
- Representation power – universal approxim
- Inductive
- Generalization, under rfitting.
- Bayesian methods for neural networks.