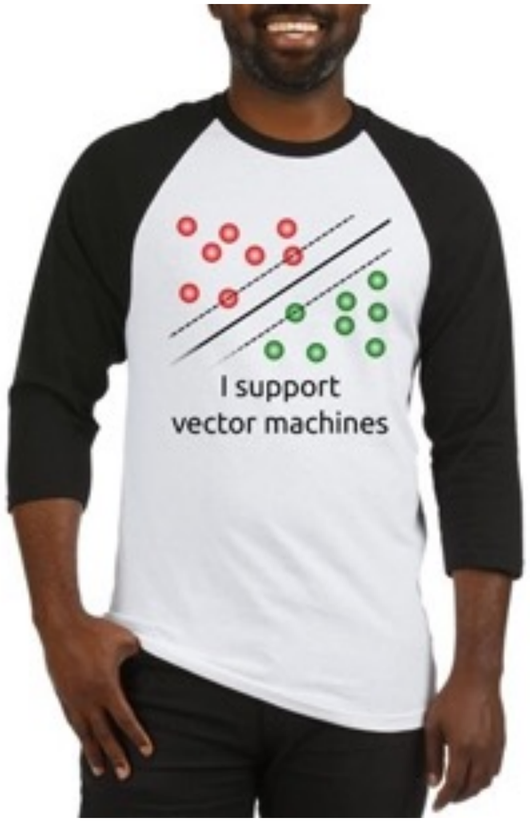
Machine Learning
CS342

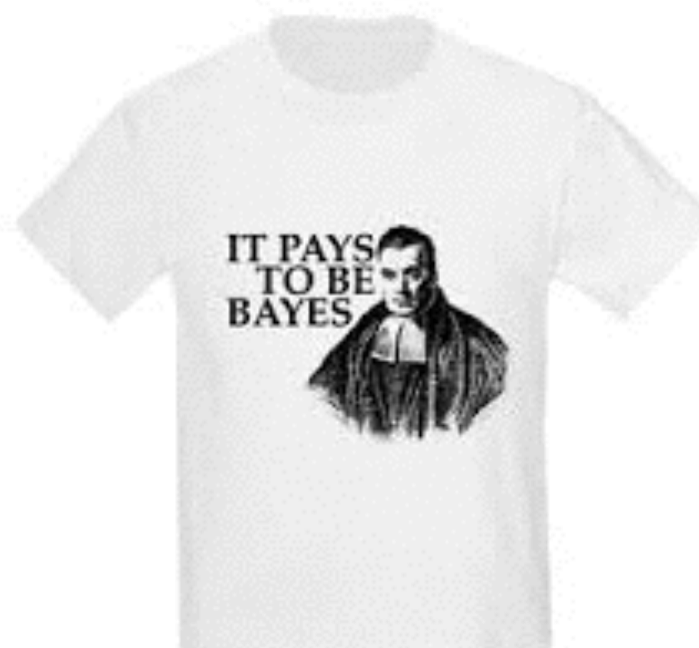
Lecture 13: Artificial Neural Networks (ANNs)

Dr. Theo Damoulas
T.Damoulas@warwick.ac.uk

Office hours: Mon & Fri 10-11am @ CS 307

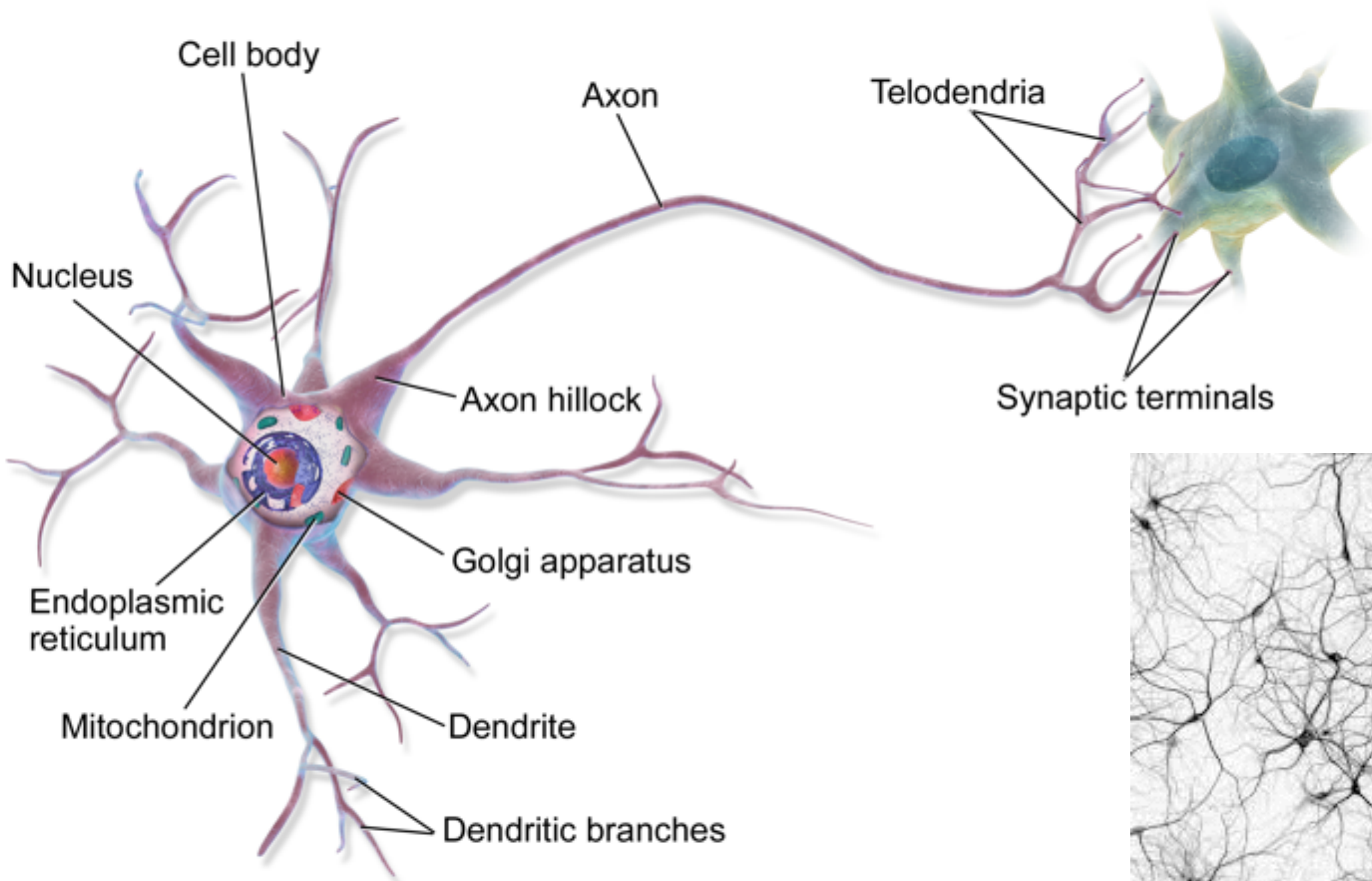
Has anyone derived the softmax from log-odds ratios?

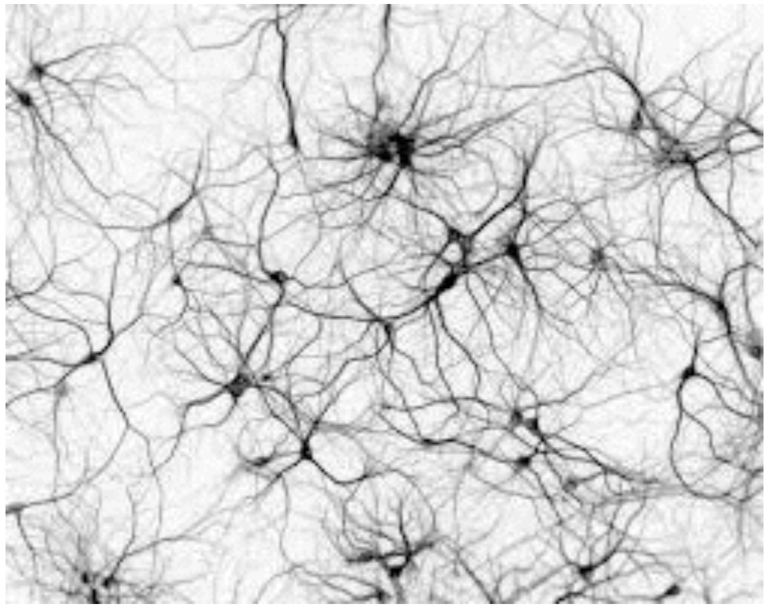




Real Neural Networks

Inspiration for ANNs as an abstraction



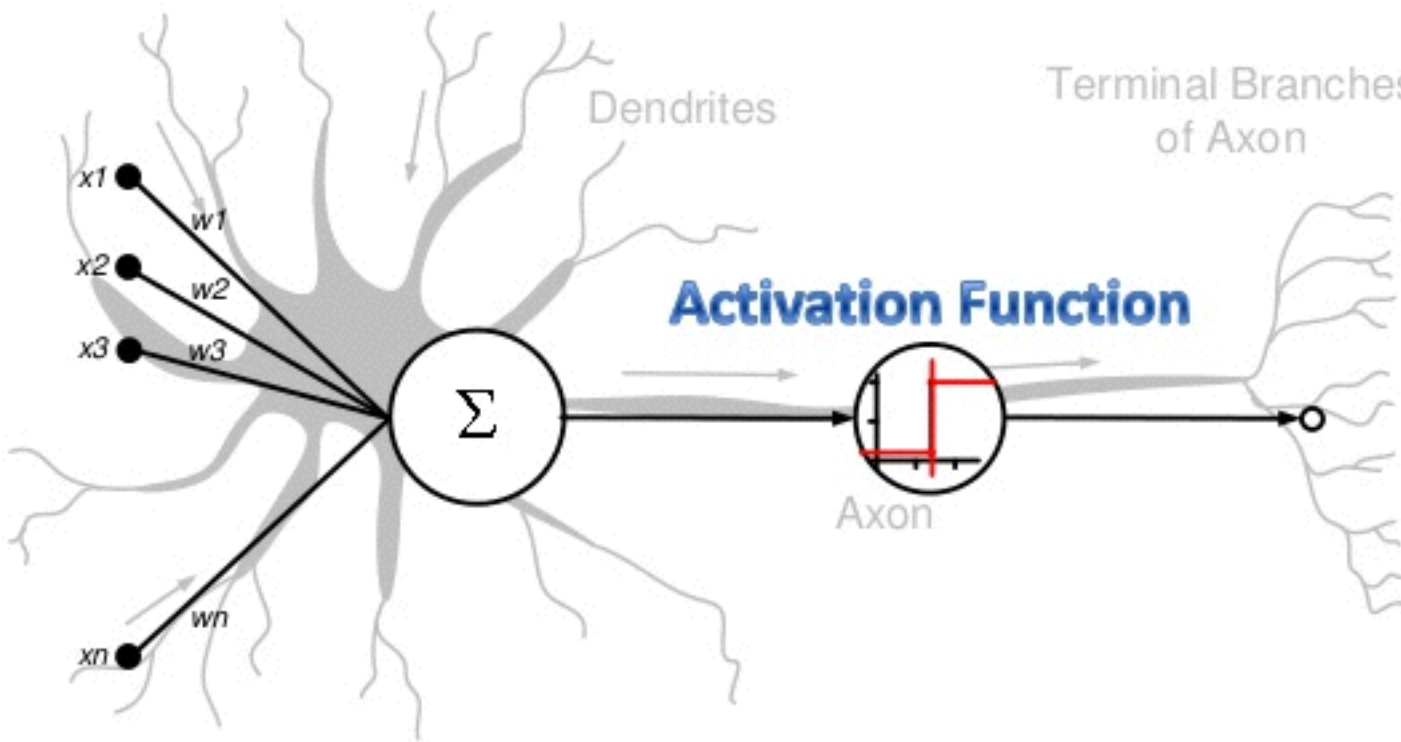


Neuroscience & Computational Neuroscience

From real NNs to ANNs

T. Mitchell book Ch. 4

Artificial Neural Networks (ANN)



- Multiple inputs
- Output to next Neuron
- “Weight” and plasticity
- Multiple processing units (sub-models)

Slide credit : Andrew L. Nelson

Artificial Neural Networks

Long history intertwined with Artificial Intelligence

"Connectionism" (AI phase)

Very active area of research in both the ANN and the NN - ANN interface

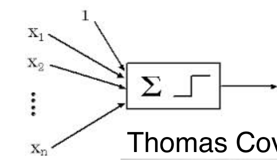
Computer science meets Statistics again

(ENIAC/Enigma) 1940 - 1950

(First learning system) 1952

Arthur Samuels

Frank Rosenblatt



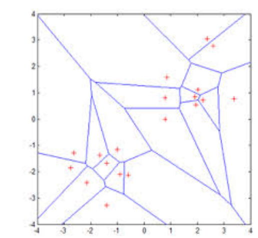
Connectionism

(Perceptron) 1957

Thomas Cover

(Machine Translation) 1966

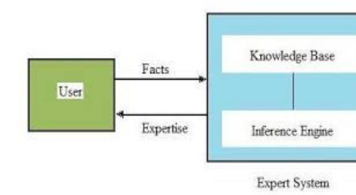
(k-NN) 1967



Symbolic

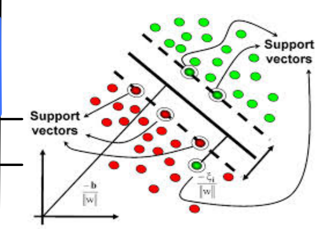
(Lighthill report) 1973

Edward Feigenbaum



Expert Systems

(LISP machines) 1987

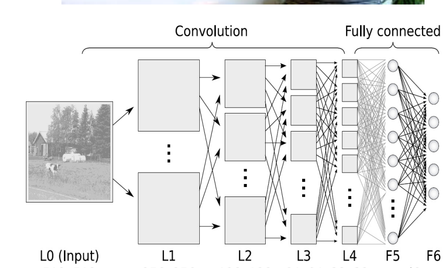


Vladimir Vapnik

Statistical ML
SVM 1993

Probabilistic ML

Geoff Hinton



OR/OPT ML

Connectionism #2 (Deep L.)

Back in time: The simplest NN: Perceptron

T. Mitchell book Ch. 4

Frank Rosenblatt

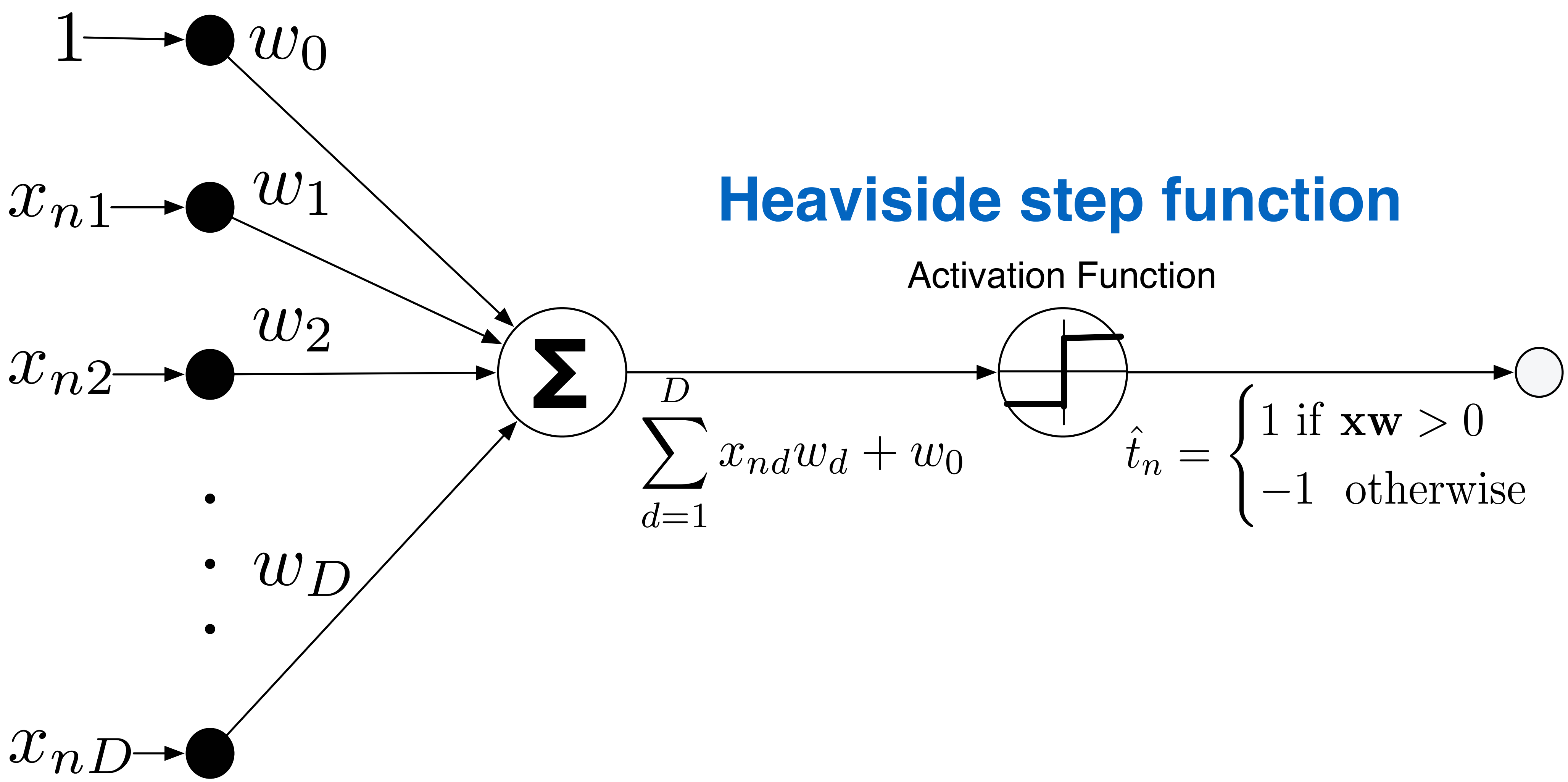
Cornell **1957**

IBM 704 computer

Mark 1 Perceptron

Perceptron

Reminds you of anything?



All of that is summarised as:

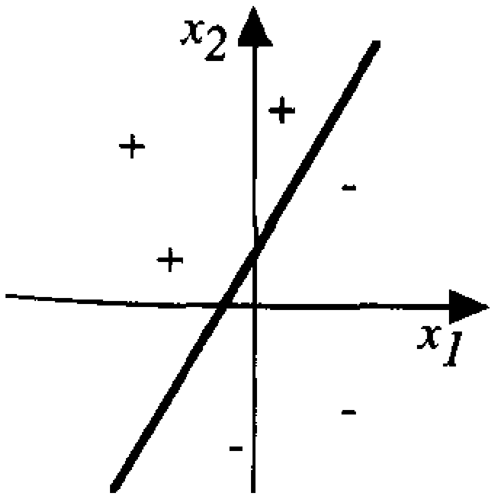
$$\boxed{\hat{t}_n = \mathrm{sgn}(\mathbf{x}_n \mathbf{w})}$$

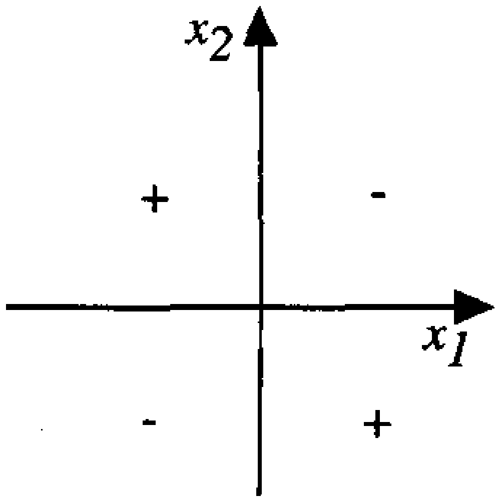
Binary classification

Training a Perceptron

As usual the learning process consists of learning the parameters

A perceptron is successful in linearly-separated problems and can model many boolean functions (AND,OR,NAND,NOR) but not XOR





Cannot model non-linearly separated data

Training a Perceptron

T. Mitchell book Ch. 4

What do we want?
Given training examples to learn w such that data correctly classified

We will do something very simple:
Perceptron training rule

$$w_d \leftarrow w_d + \Delta w_d$$

$$\Delta w_d = \eta(t - \hat{t})x_{nd}$$

Notation differences
(we use d for attributes)

where η is a positive constant called the *learning rate* and controls how fast the parameters change (e.g. 0.1)

If you do some examples with +1/-1 examples you will see it makes sense

Training a Perceptron

If problem non-linearly separable this will not converge (Error not 0)

```
Initialise w randomly
eta = 0.1 (for example);
while there is a non-zero error
  for i = 1 to N (number of training examples)
     Choose ith training example x,t
     Compute dot product xw
     Compute error(i) = t-sign(xw)
     Update w += eta*error(i)*xT
```

Convince yourself of the equivalence

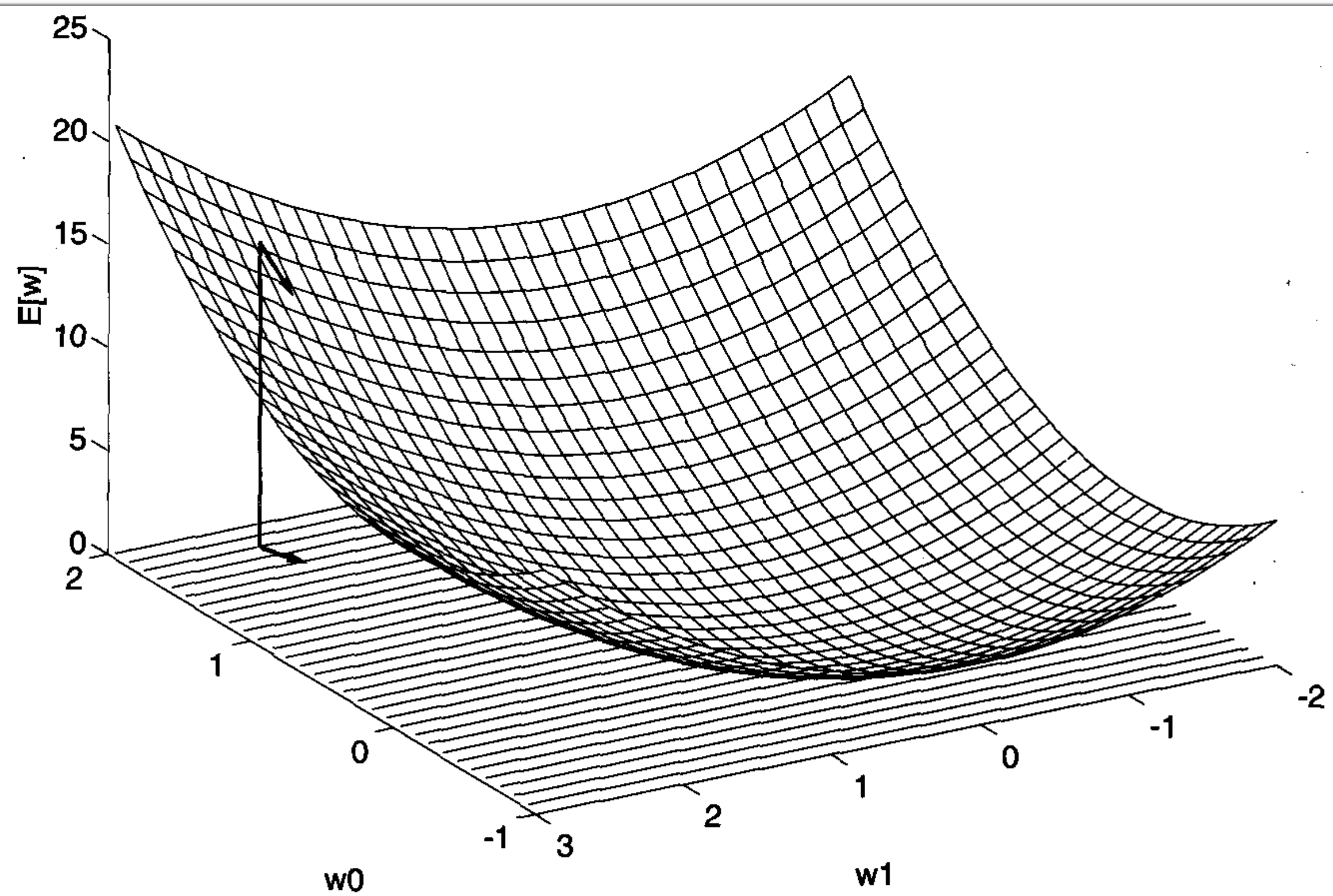
$$w_d \leftarrow w_d + \Delta w_d$$

$$\Delta w_d = \eta(t - \hat{t})x_{nd}$$

Gradient Descent and the Delta rule

Our previous algorithm will not converge if data is not linearly separable
We need a way to find convergence towards best-fit solution in that case

Gradient descent: *We want to minimise the error with respect to our parameters so lets change our parameters in the direction of steepest descent in the error surface*



Gradient Descent and the Delta rule

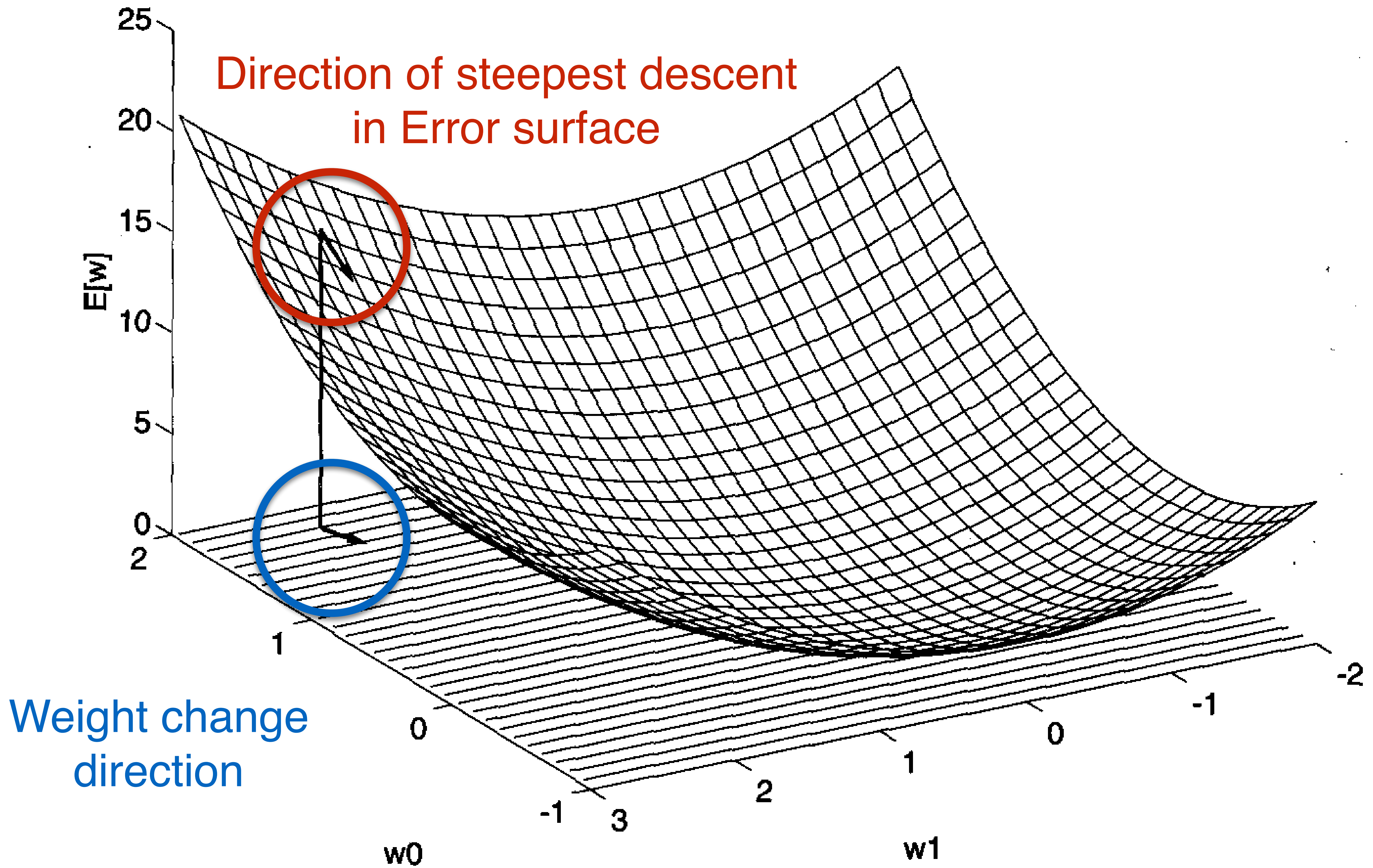
Gradient descent: *We want to minimise the error with respect to our parameters so lets change our parameters in the direction of* ***steepest descent in the error surface***

Gradient information!

$$\nabla E(\mathbf{w}) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_D}\right]$$

A vector in weight space that specifies direction of steepest increase in E
So we negate this to find direction of steepest decrease!

Gradient Descent and the Delta rule



Gradient Descent and the Delta rule

So here is our new weight updates:

$$w_d \leftarrow w_d - \eta \frac{\partial E}{\partial w_d}$$

All together (vector-matrix format):

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla E(\mathbf{w})$$

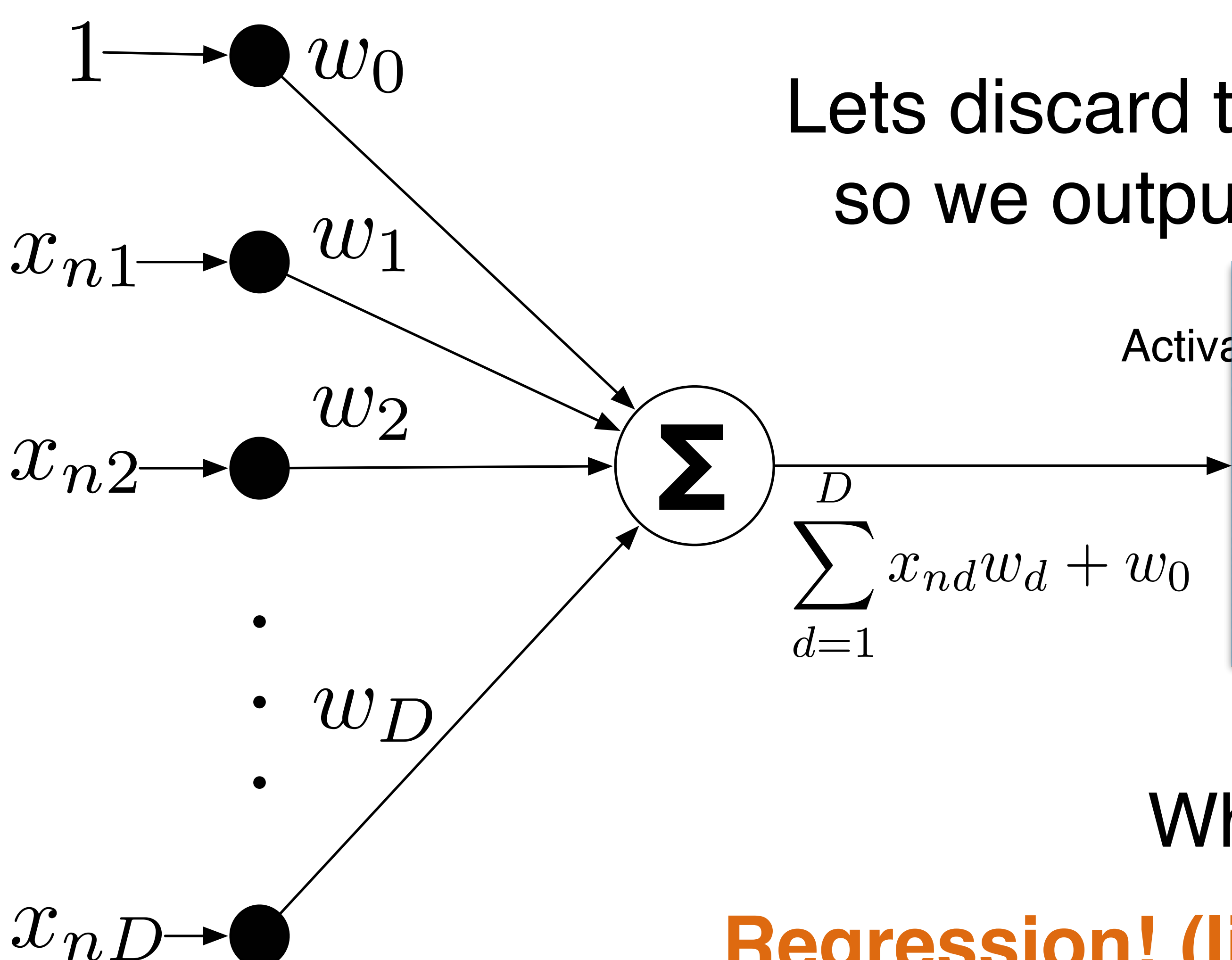
So we are changing every component of **w** in proportion to its derivative

So to get our algorithm we only need a way to estimate these derivatives at every iteration

Gradient Descent on linear unit perceptron

$$\boxed{\hat{t}_n = \mathrm{sgn}(\mathbf{x}_n \mathbf{w})}$$

I defined the error as (t_n - sgn($\mathbf{x}_n\mathbf{w}$)) and differentiating that wrt **w** is not nice



Lets discard the activation function so we output the continuous **xw**

What is this?

Regression! (linear unit perceptron)

Gradient Descent on linear unit perceptron

Regression! (linear unit perceptron)

So what error do we know for regression?

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}(t_n - \mathbf{x}_n\mathbf{w})^2$$

Just some constant different from SQE Loss so won't affect really

$$\frac{\partial E}{\partial \mathbf{w}} = -\sum_{n=1}^{N}(t_n - \mathbf{x}_n\mathbf{w})\mathbf{x}_n^{\mathrm{T}}$$

So the update is:

$$\boxed{\mathbf{w} \leftarrow \mathbf{w} + \eta\sum_{n=1}^{N}(t_n - \mathbf{x}_n\mathbf{w})\mathbf{x}_n^{\mathrm{T}}}$$

Gradient Descent (Batch-mode) on linear-unit perceptron

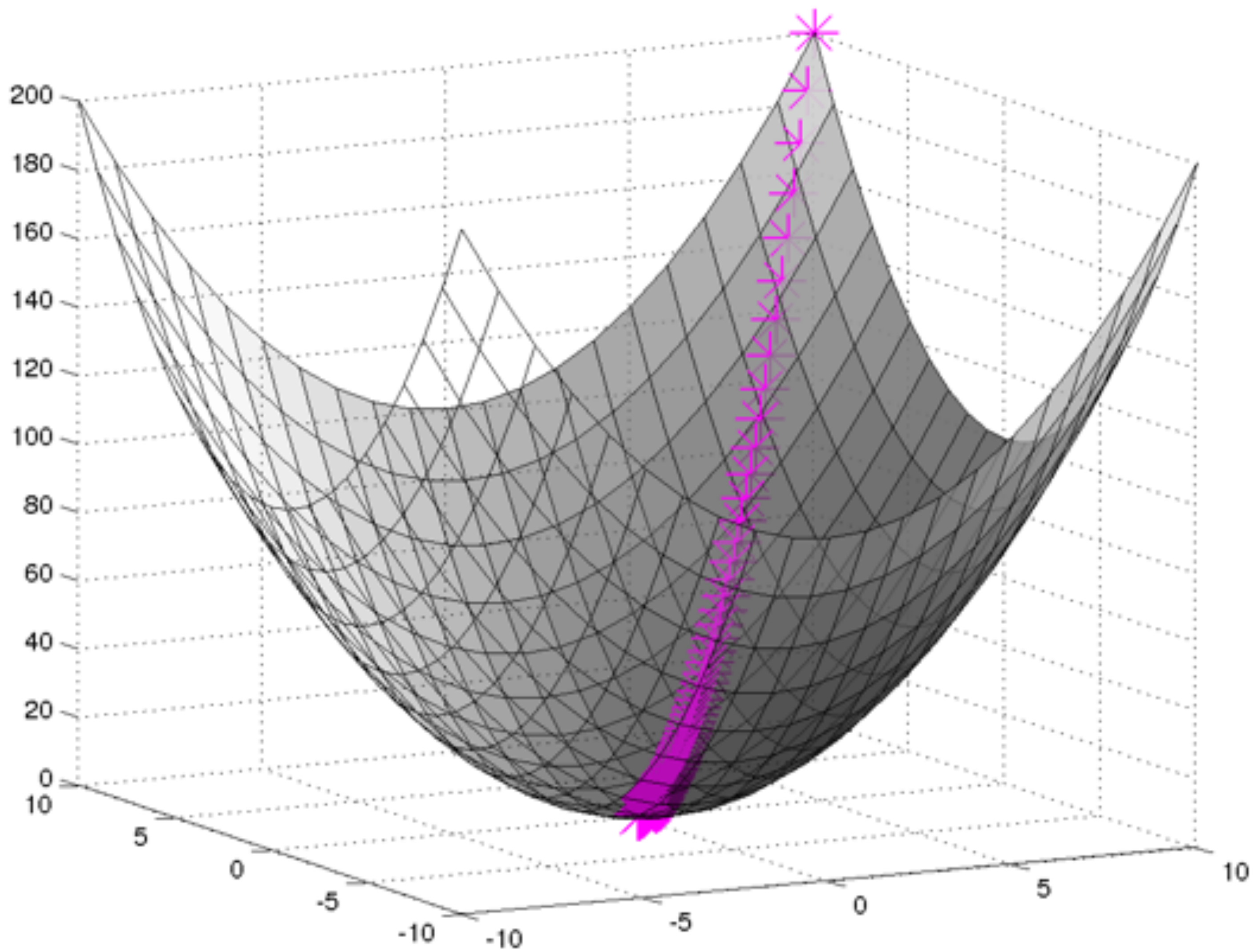
We are finding the OLS solution with a NN and gradient descent!

```
Initialise w randomly
eta = 0.1;
while not converged
     Update w += eta*X^T(t - Xw)
```

Notice we are using all the data **t**,**X** for the update each time!

But I already know the exact solution to minimise this error/Loss
Just set the derivative to 0 to get OLS solution

Gradient Descent (Batch-mode) on linear-unit perceptron



In this case (linear unit) global minimum same as OLS solution

Stochastic Gradient Descent

We can actually perform incremental gradient descent

Like the perceptron algorithm, *update weights after every training example*

Incremental/Stochastic Gradient Descent

```
Initialise w randomly
eta = 0.01; (typically smaller then batch mode)
while not converged
  for i = 1 to N (number of training examples)
     Choose ith training example x,t
     Update w += eta*(t - xw)xT
```

Notice we are using only one observation **x**,t for the update each time!

It helps overcome local minima in some cases!

Differences between Batch-mode and Stochastic GD

- In standard (batch-mode) gradient descent the error is summed over all examples before updating weights, whereas in stochastic gradient descent weights are updated upon examining each training example
- More computation per weight update for batch-mode gradient descent.
- Batch-mode gradient descent uses the true gradient so we make larger steps (larger learning rate eta) then the stochastic version
- In cases where there are multiple local minima in the global error function stochastic gradient descent can sometimes avoid falling into these as it uses the gradient of the error with respect a single training example

Perceptron training rule versus Delta training rule

Today!

- Two algorithms for iteratively learning perceptron weights:
  - Perceptron training (linearly separable)
  - (Stochastic and Batch-mode) Gradient Descent with Delta training rule (linearly and non-linearly separable)

We have also seen the following perceptron cases:

- Heaviside step activation function used for binary classification
- The linear unit (unthresholded perceptron) used for linear regression

Any other activation function?

ANNs in general

Next Friday: Multilayer Perceptrons (MLPs) and Back-propagation

An appropriate model in these situations:

- Big Data: Modern ANNs require massive amounts of data to fit the typically thousands of parameters. Especially in "Deep Learning"
- Target output can be discrete-valued (e.g. classification), continuous-valued (e.g. regression), or a vector of either type (multi-output)
- Can handle noise and errors
- Long training times as compared to other models (e.g. DTs)