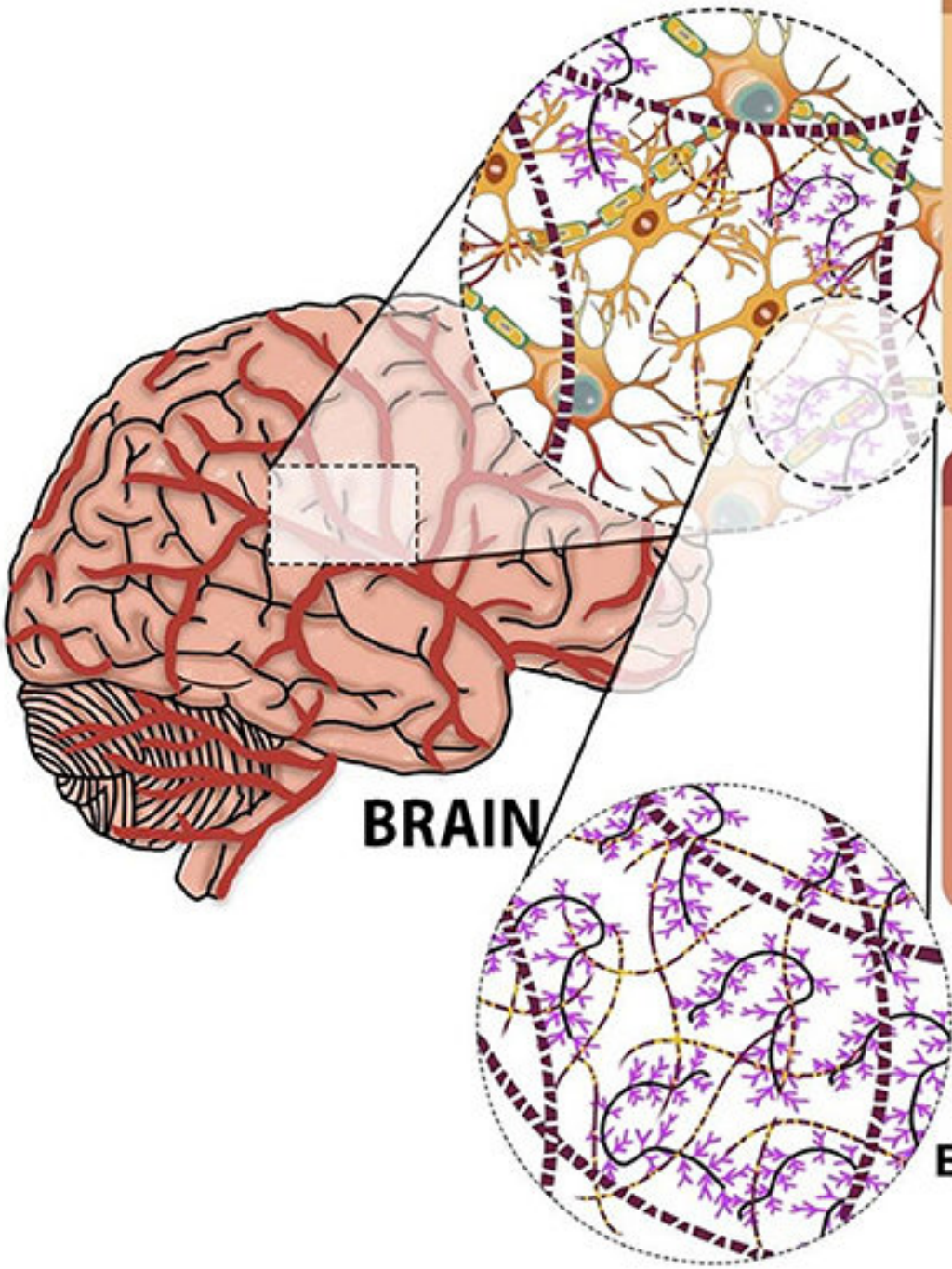
THE PERCEPTRON



Brain Cell ID

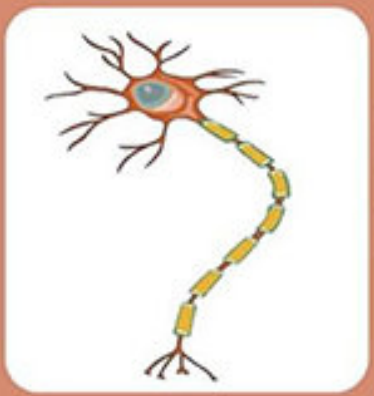
Name Glial cell

Residency Brain

Occupation

- Main immune cell of the brain.
- Helps protect the brain from infection and cellular damage.

Brain Cell ID

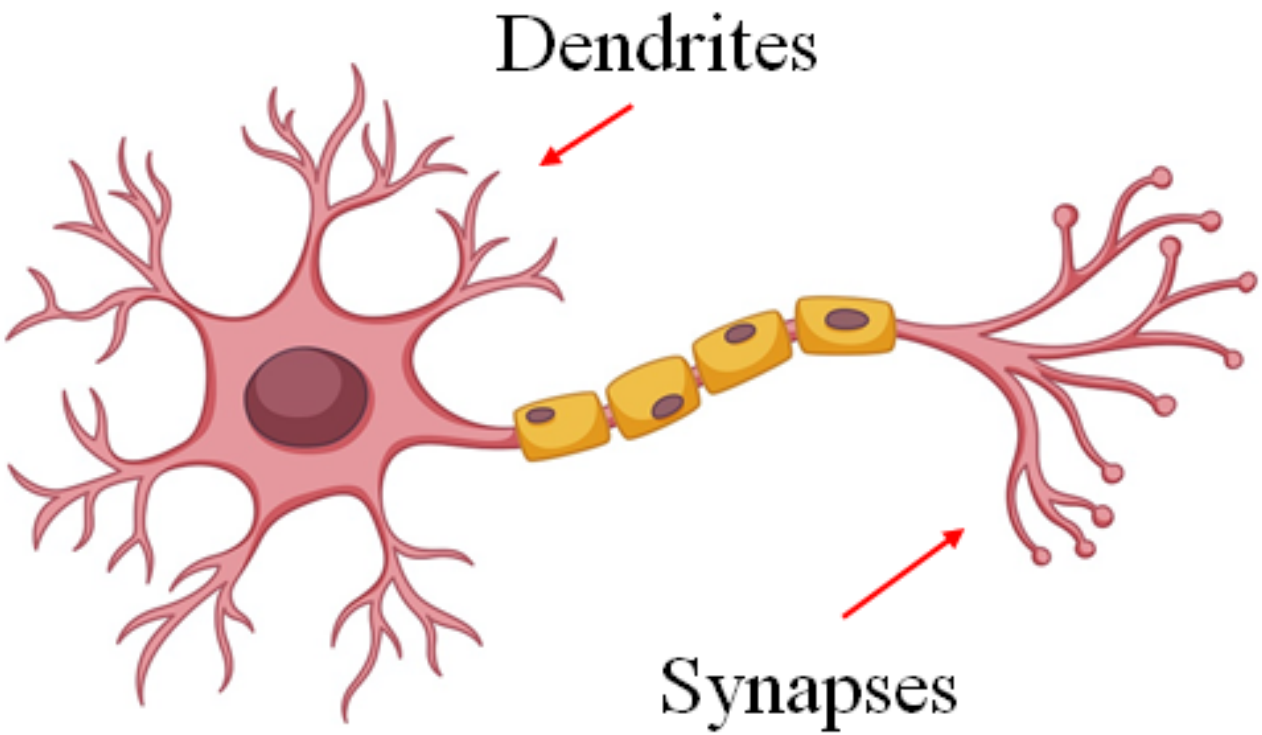


Name Neuron

Residency Brain

Occupation

-Basic unit of the nervous system.
-Functions to process and transmit information to other neurons.

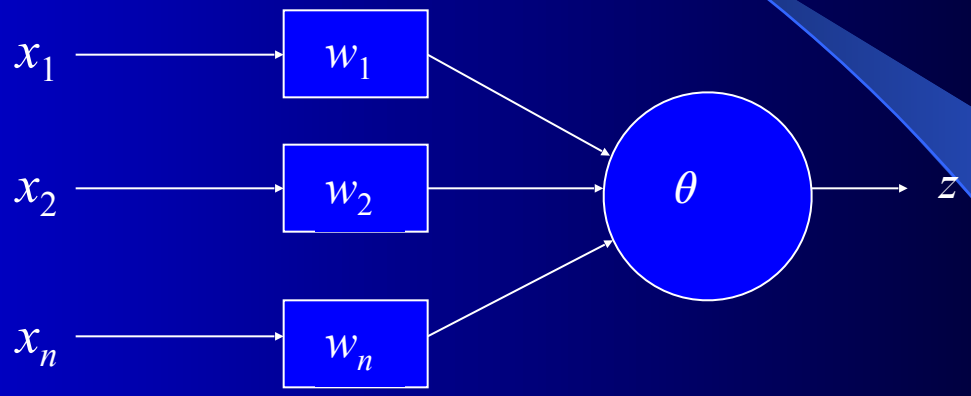


NEURON

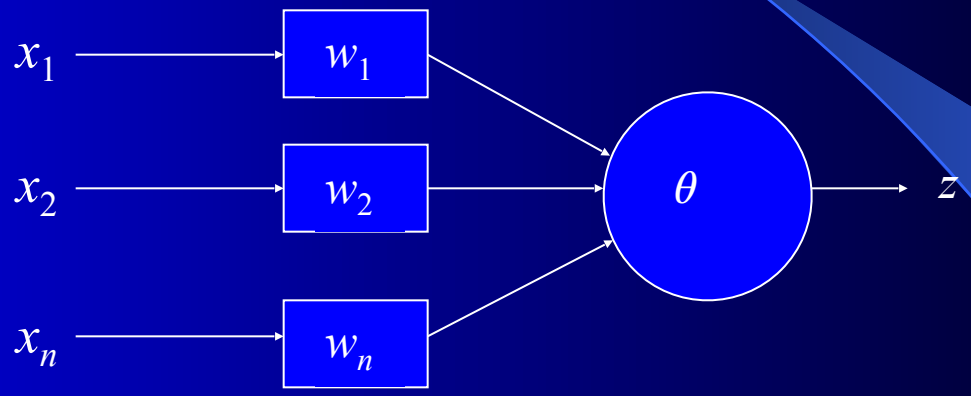
Perceptron Learning Algorithm

- First neural network learning model in the 1960's
  - Frank Rosenblatt
- Simple and limited (single layer model)
- Basic concepts are similar for multi-layer models so this is a good learning tool
- Still used in some current applications (large business problems, where intelligibility is needed, etc.)

Perceptron Node – Threshold Logic Unit



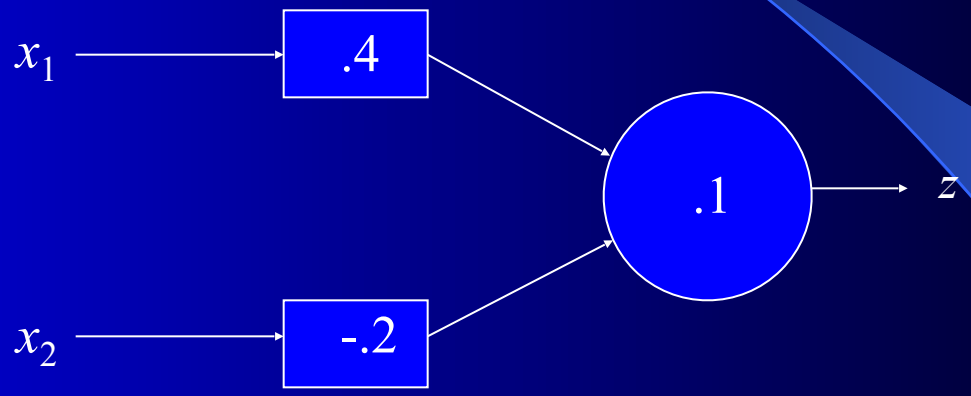
Perceptron Node – Threshold Logic Unit



- Learn weights such that an objective function is maximized.

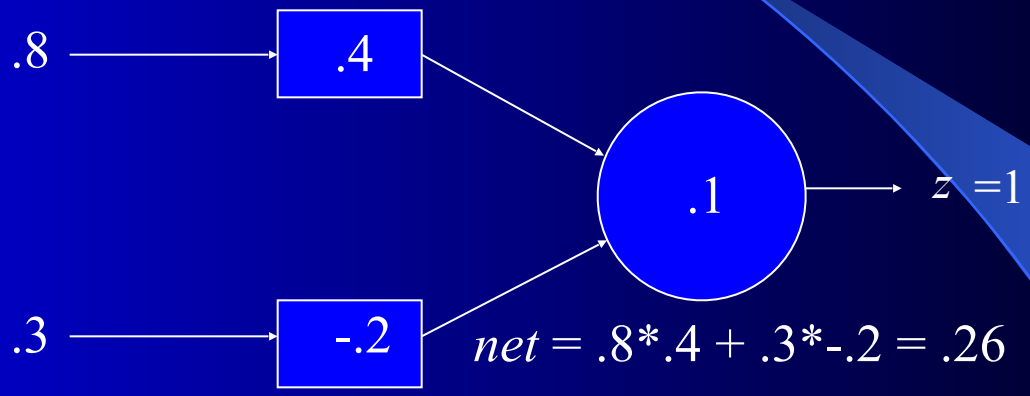
- What objective function should we use?

Perceptron Learning Algorithm



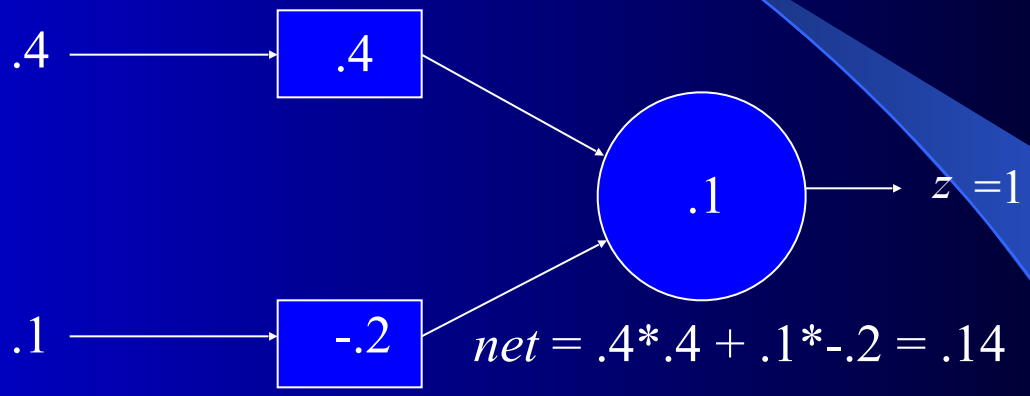
x_1	x_2	t
.8	.3	1
.4	.1	0

First Training Instance



x_1	x_2	t
.8	.3	1
.4	.1	0

Second Training Instance



x_1	x_2	t
.8	.3	1
.4	.1	0

Need to make a correction

Perceptron Learning Rule

$$\Delta w_i = c(t - z)\, x_i$$

- Where w_i is the weight from input i to the perceptron node,
  - c is the learning rate,
  - t is the target for the current instance,
  - z is the current output, and
  - x_i is i^{th} input

- Least perturbation principle
  - Only change weights if there is an error
  - small c rather than changing weights sufficient to make current instance correct
  - Scale by x_i

$$w_{ij} \leftarrow w_{ij} - \eta(y_j - t_j) \cdot x_i$$

Augmented Data Instances

1 0 1 -> 0
1 0 0 -> 1

Augmented Version
1 0 1 1 -> 0
1 0 0 1 -> 1

- Treat threshold like any other weight. No special case. Call it a *bias* since it biases the output up or down.
- Since we start with random weights anyways, can ignore the -θ notion, and just think of the bias as an extra available weight. (note the author uses a -1 input)
- Always use a bias weight

Perceptron Learning Rule

- Create a perceptron node with *n* inputs
- Iteratively select a data instance from the training set
- Calculate the output value z
- Apply the perceptron rule to adjust weights
- Each iteration through the **training set** is an *epoch*
- Continue training until total training set error ceases to improve (maybe)
- Perceptron Convergence Theorem: Guaranteed to find a solution in finite time if a solution exists

Perceptron Rule Example

- Assume a 3 input perceptron plus bias (it outputs 1 if net > 0, else 0)
- Assume a learning rate c of 1 and initial weights all 0: $\Delta w_i = c(t - z)\, x_i$
- Training set
  0 0 1 -> 0
  1 1 1 -> 1
  1 0 1 -> 1
  0 1 1 -> 0

Data	Target (t)	Weight Vector (w_i)	Net	Output (z)	ΔW
0 0 1 1	0	0 0 0 0			

Example

- Assume a 3 input perceptron plus bias (it outputs 1 if net > 0, else 0)
- Assume a learning rate c of 1 and initial weights all 0: $\Delta w_i = c(t - z)\, x_i$
- Training set
  0 0 1 -> 0
  1 1 1 -> 1
  1 0 1 -> 1
  0 1 1 -> 0

Data	Target (t)	Weight Vector (w_i)	Net	Output (z)	ΔW
0 0 1 1	0	0 0 0 0	0	0	0 0 0 0
1 1 1 1	1	0 0 0 0			

Example

- Assume a 3 input perceptron plus bias (it outputs 1 if net > 0, else 0)
- Assume a learning rate c of 1 and initial weights all 0: $\Delta w_i = c(t - z)\, x_i$
- Training set
  0 0 1 -> 0
  1 1 1 -> 1
  1 0 1 -> 1
  0 1 1 -> 0

Data	Target (t)	Weight Vector (w_i)	Net	Output (z)	ΔW
0 0 1 1	0	0 0 0 0	0	0	0 0 0 0
1 1 1 1	1	0 0 0 0	0	0	1 1 1 1
1 0 1 1	1	1 1 1 1			

Challenge Question - Perceptron

- Assume a 3 input perceptron plus bias (it outputs 1 if net > 0, else 0)
- Assume a learning rate c of 1 and initial weights all 0: $\Delta w_i = c(t - z)\, x_i$
- Training set
  0 0 1 -> 0
  1 1 1 -> 1
  1 0 1 -> 1
  0 1 1 -> 0

Data	Target (t)	Weight Vector (w_i)	Net	Output (z)	ΔW
0 0 1 1	0	0 0 0 0	0	0	0 0 0 0
1 1 1 1	1	0 0 0 0	0	0	1 1 1 1
1 0 1 1	1	1 1 1 1			

- Once it converges the final weight vector will be
  A. 1 1 1 1
  B. -1 0 1 0
  C. 0 0 0 0
  D. 1 0 0 0
  E. None of the above

Example

- Assume a 3 input perceptron plus bias (it outputs 1 if net > 0, else 0)
- Assume a learning rate c of 1 and initial weights all 0: $\Delta w_i = c(t - z)\, x_i$
- Training set
  0 0 1 -> 0
  1 1 1 -> 1
  1 0 1 -> 1
  0 1 1 -> 0

Data	Target (t)	Weight Vector (w_i)	Net	Output (z)	ΔW
0 0 1 1	0	0 0 0 0	0	0	0 0 0 0
1 1 1 1	1	0 0 0 0	0	0	1 1 1 1
1 0 1 1	1	1 1 1 1	3	1	0 0 0 0
0 1 1 1	0	1 1 1 1			

Example

- Assume a 3 input perceptron plus bias (it outputs 1 if net > 0, else 0)
- Assume a learning rate c of 1 and initial weights all 0: $\Delta w_i = c(t - z)\, x_i$
- Training set
  0 0 1 -> 0
  1 1 1 -> 1
  1 0 1 -> 1
  0 1 1 -> 0

Data	Target (t)	Weight Vector (w_i)	Net	Output (z)	ΔW
0 0 1 1	0	0 0 0 0	0	0	0 0 0 0
1 1 1 1	1	0 0 0 0	0	0	1 1 1 1
1 0 1 1	1	1 1 1 1	3	1	0 0 0 0
0 1 1 1	0	1 1 1 1	3	1	0 -1 -1 -1
0 0 1 1	0	1 0 0 0			

Example

- Assume a 3 input perceptron plus bias (it outputs 1 if net > 0, else 0)
- Assume a learning rate c of 1 and initial weights all 0: $\Delta w_i = c(t - z)\, x_i$
- Training set
  0 0 1 -> 0
  1 1 1 -> 1
  1 0 1 -> 1
  0 1 1 -> 0

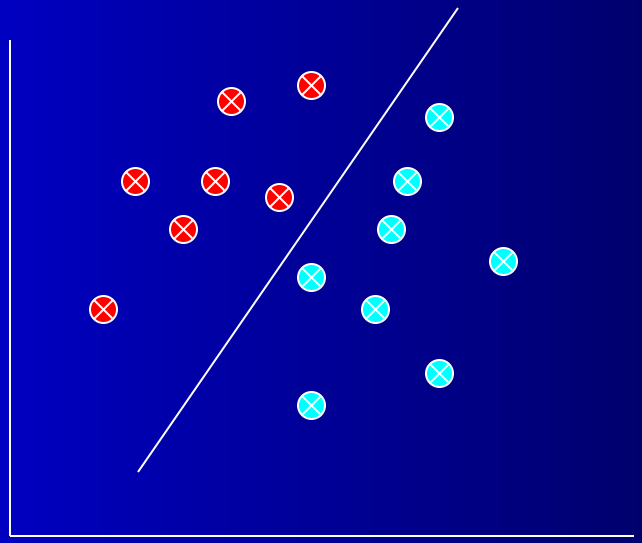
Data	Target (t)	Weight Vector (w_i)	Net	Output (z)	ΔW
0 0 1 1	0	0 0 0 0	0	0	0 0 0 0
1 1 1 1	1	0 0 0 0	0	0	1 1 1 1
1 0 1 1	1	1 1 1 1	3	1	0 0 0 0
0 1 1 1	0	1 1 1 1	3	1	0 -1 -1 -1
0 0 1 1	0	1 0 0 0	0	0	0 0 0 0
1 1 1 1	1	1 0 0 0	1	1	0 0 0 0
1 0 1 1	1	1 0 0 0	1	1	0 0 0 0
0 1 1 1	0	1 0 0 0	0	0	0 0 0 0

Perceptron Homework

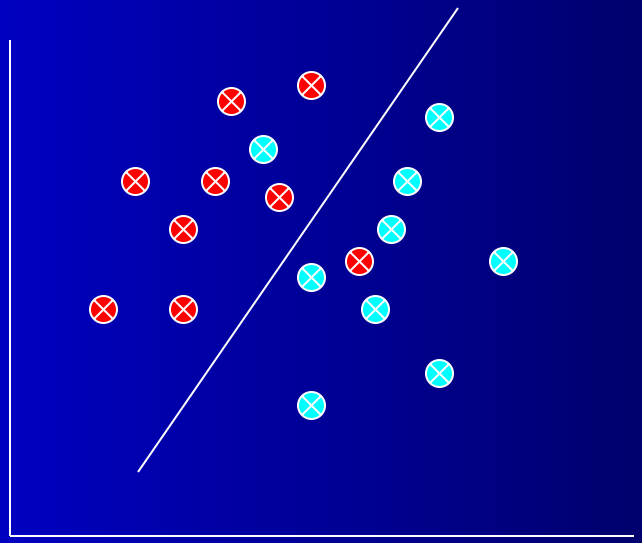
- Assume a 3 input perceptron plus bias (it outputs 1 if net > 0, else 0)
- Assume a learning rate c of 1 and initial weights all 1: $\Delta w_i = c(t - z)\, x_i$
- Show weights after each instance for just one epoch
- Training set
  1 0 1 -> 0
  1 .5 0 -> 0
  1 -.4 1 -> 1
  0 1 .5 -> 1

Data	Target (t)	Weight Vector (w_i)	Net	Output (z)	ΔW
		1 1 1 1			

Linear Separability

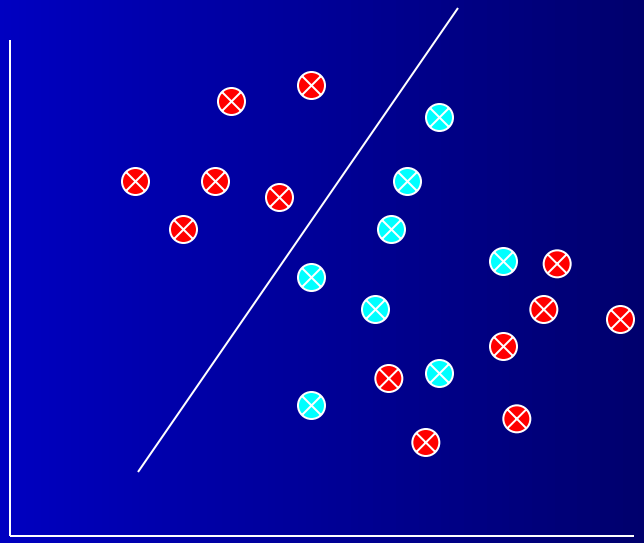


Linear Separability and Generalization



When is data noise vs. a legitimate exception

Limited Functionality of Hyperplane



FEATURE ENGINEERING

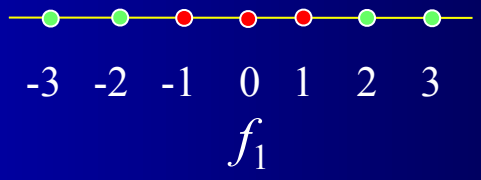
Linear Models which are Non-Linear in the Input Space

- We could preprocess the inputs in a non-linear way
- To the perceptron algorithm is the same but with more/different inputs. It still uses the same update rules
- For example, for a problem with two inputs x and y (plus the bias), we could also add the inputs x^2, y^2, and $x \cdot y$
- The perceptron would just think it is a 5-dimensional task, and it is linear (5-d hyperplane) in those 5 dimensions
  - But what kind of decision surfaces would it allow for the original 2-*d* input space?

Quadric Machine

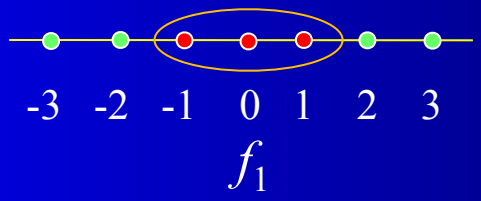
- All quadratic surfaces (2nd order)
  - ellipsoid
  - parabola
  - etc.
- That significantly increases the number of problems that can be solved
- But still many problem which are not quadratically separable
- Could go to 3rd and higher order features, but number of possible features grows exponentially
- Multi-layer neural networks will allow us to discover high-order features automatically from the input space

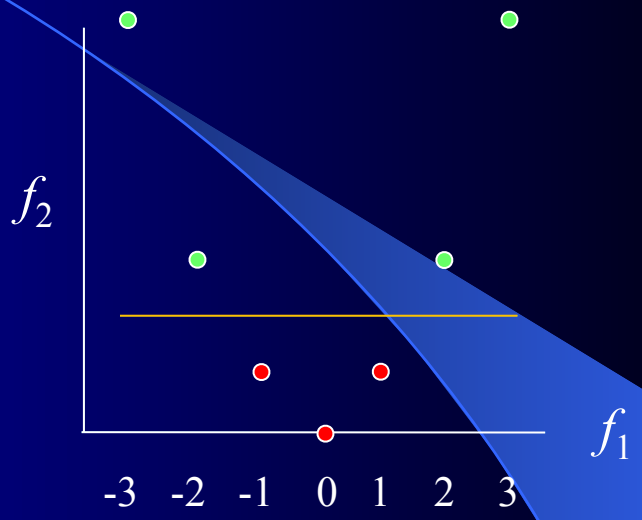
Simple Quadric Example



- What is the decision surface for a 1-d (1 input) problem?
- Perceptron with just feature f_1 cannot separate the data
- Could we add a transformed feature to our perceptron?

Simple Quadric Example





- Perceptron with just feature f_1 cannot separate the data
- Could we add another feature to our perceptron $f_2 = f_1^2$

Quadric Machine Homework

- Assume a 2-input perceptron expanded to be a quadric (2nd order) perceptron, with 5 input weights (x, y, $x \cdot y$, x^2, y^2) and the bias weight
  - Assume it outputs 1 if net > 0, else 0
- Assume a learning rate c of .5 and initial weights all 0
  - $\Delta w_i = c(t - z)\, x_i$
- Show all weights after each instance for one epoch with the following training set

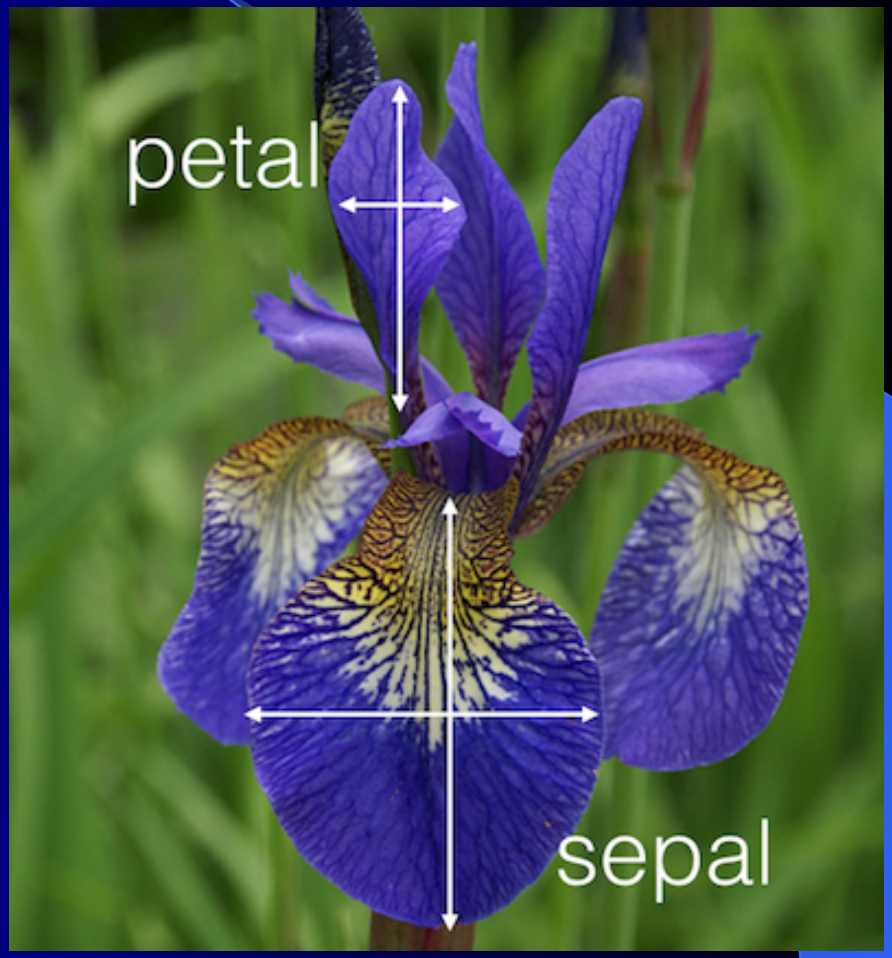
x	y	Target
0	.4	0
-.1	1.2	1
.5	.8	0

How to Handle Multi-Class Output

- This is an issue with learning models which only support binary classification (perceptron, SVM, etc.)
- Create 1 perceptron for each output class, where the training set considers all other classes to be negative examples (one vs the rest)
  - Run all perceptrons on novel data and set the output to the class of the perceptron which outputs high
  - If there is a tie, choose the perceptron with the highest net value

UC Irvine Machine Learning Data Base
Iris Data Set

4.8,3.0,1.4,0.3,	Iris-setosa
5.1,3.8,1.6,0.2,	Iris-setosa
4.6,3.2,1.4,0.2,	Iris-setosa
5.3,3.7,1.5,0.2,	Iris-setosa
5.0,3.3,1.4,0.2,	Iris-setosa
7.0,3.2,4.7,1.4,	Iris-versicolor
6.4,3.2,4.5,1.5,	Iris-versicolor
6.9,3.1,4.9,1.5,	Iris-versicolor
5.5,2.3,4.0,1.3,	Iris-versicolor
6.5,2.8,4.6,1.5,	Iris-versicolor
6.0,2.2,5.0,1.5,	Iris-viginica
6.9,3.2,5.7,2.3,	Iris-viginica
5.6,2.8,4.9,2.0,	Iris-viginica
7.7,2.8,6.7,2.0,	Iris-viginica
6.3,2.7,4.9,1.8,	Iris-viginica



Determining Model Performance

Objective Functions: Accuracy

- How do we judge the quality of a particular model (e.g. Perceptron with a particular setting of weights)
- Consider how accurate the model is on the data set
  - *Classification accuracy* = # Correct/Total instances
  - *Classification error* = # Misclassified/Total instances (= 1 – acc)

Objective Functions: Error

- Usually minimize a Loss function (aka cost, error)
- For real valued outputs and/or targets
  - Instance error = Target – output: Errors could cancel each other
    - $\Sigma|t_j - z_j|$ (L1 loss), where j indexes all outputs in the instance
    - Common approach is *Squared Error* = $\Sigma(t_j - z_j)^2$ (L2 loss)
- For nominal data, error is typically 1 for a mismatch and 0 for a match
  - For nominal (including binary) output and targets, L1, L2, and classification error are equivalent

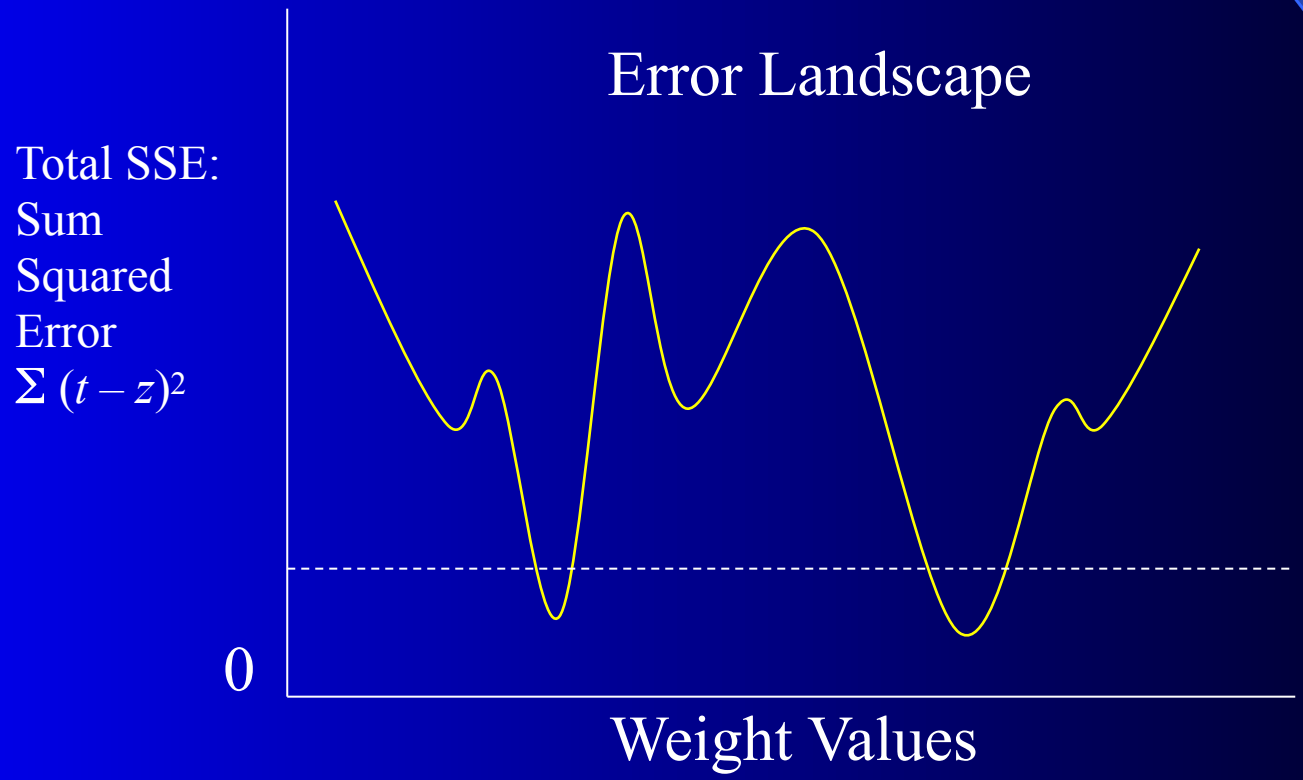
Mean Squared Error

- Mean Squared Error (MSE) = SSE/n where n is the number of instances in the data set
  - This can be nice because it normalizes the error for data sets of different sizes
  - MSE is the average squared error per instance

- Root Mean Squared Error (RMSE) – is the square root of the MSE
  - This puts the error value back into the same units as the features and can thus be more intuitive
    - Since we squared the error on the SSE
  - RMSE is the average distance (error) of targets from the outputs in the same scale as the features
  - Note RMSE is the square-root of the total data set MSE, and NOT the sum of the root of each individual instance MSE

Error Surface

- Error is a function of the weights
  - $E = \Sigma(t_i - z_i)^2 = \Sigma(t_i - \Sigma x_j w_{ij})^2$
- If we could search this space, we could find the minimum



Challenge Question - Error

- Given the following data set, what is the L1 ($\Sigma|t_i - z_i|$), SSE (L2) ($\Sigma(t_i - z_i)^2$), MSE, and RMSE error for the entire data set?

x	y	Output	Target	Data Set
2	-3	1	1	
0	1	0	1	
.5	.6	.8	.2	
L1				?
SSE				?
MSE				?
RMSE				?

A. .4 1 1 1
B. 1.6 2.36 1 1
C. .4 .64 .21 0.453
D. 1.6 1.36 .453 .673
E. None of the above

Challenge Question - Error

- Given the following data set, what is the L1 ($\Sigma|t_i - z_i|$), SSE (L2) ($\Sigma(t_i - z_i)^2$), MSE, and RMSE error for the entire data set?

x	y	Output	Target	Data Set
2	-3	1	1	
0	1	0	1	
.5	.6	.8	.2	
L1	0	1	0.6	1.6
SSE	0	1	0.36	1.36
MSE				1.36/3 = .453
RMSE				.45^.5 = .67

A. .4 1 1 1
B. 1.6 2.36 1 1
C. .4 .64 .21 0.453
D. 1.6 1.36 .453 .673
E. None of the above

Error Values Homework

- Given the following data set, what is the L1, SSE (L2), MSE, and RMSE error?

Instance	x	y	Output	Target	Data Set
1	-1	-1	.6	1.0	
2	-1	1	-.3	0	
3	1	-1	1.2	.5	
4	1	1	0	-.2	
L1			?		?
SSE			?		?
MSE			?		?
RMSE			?		?

Gradient Descent Learning: Minimize (Maximize) the Objective Function

- Gradient descent algorithm
  - Find a starting location – set of weights
  - Loop
    - Calculate output values
    - Use the gradient to adjust your weight values
- Adjusting the weights
  - Derivative of the error function w.r.t the weights – slope or gradient

$$w_i \leftarrow w_i + \Delta w_i$$

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

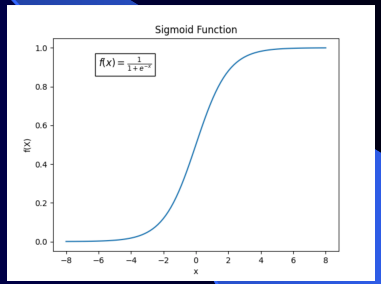
Deriving a Gradient Descent Learning Algorithm

- Goal is to decrease overall error (or other loss function) each time a weight is changed
- Sum Squared error one possible loss function $E = \Sigma (t - z)^2$
  - Actually use $E = \frac{1}{2}\Sigma (t - z)^2$
- Other reasons to use SSE
  - All errors are positive
  - Amplifies the effect of larger errors
  - Transforms the error surface – smooth and differentiable
- Partial derivative of the error function w.r.t the weights gives us a weight update function

Delta rule algorithm

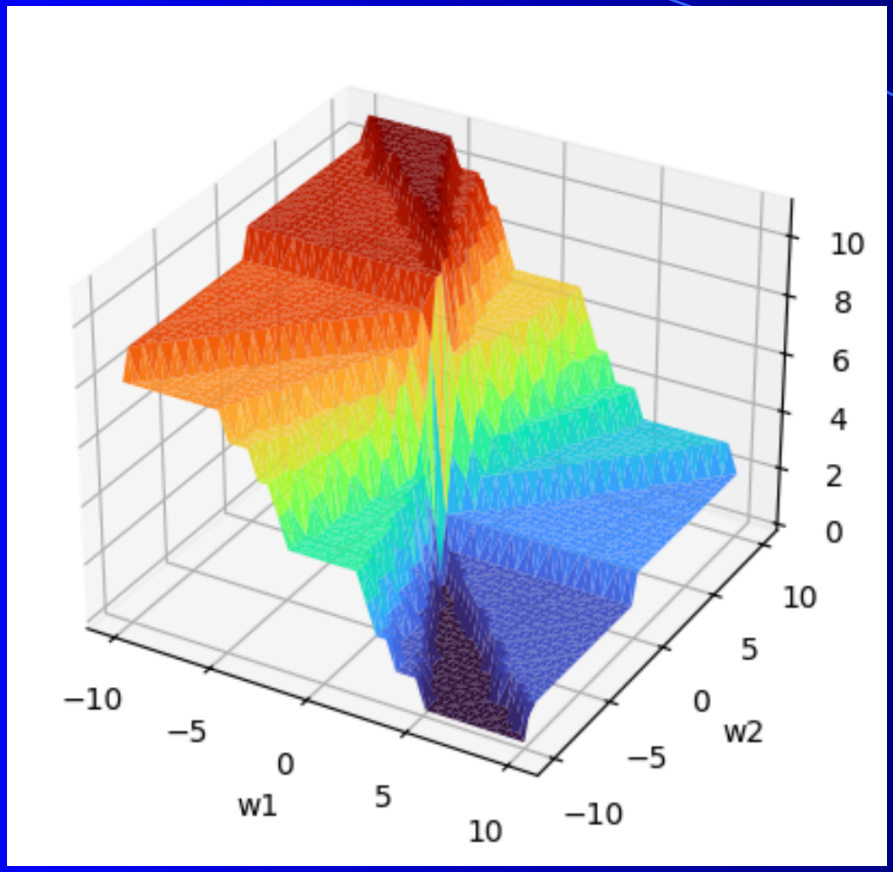
- Simple perceptron rule has a problem for gradient descent
  - Threshold output makes the error function infinite or zero everywhere.
- Delta rule uses (target - net) before the net value goes through the threshold in the learning rule to decide weight update

Use sigmoid(net)
if you want 0/1 output

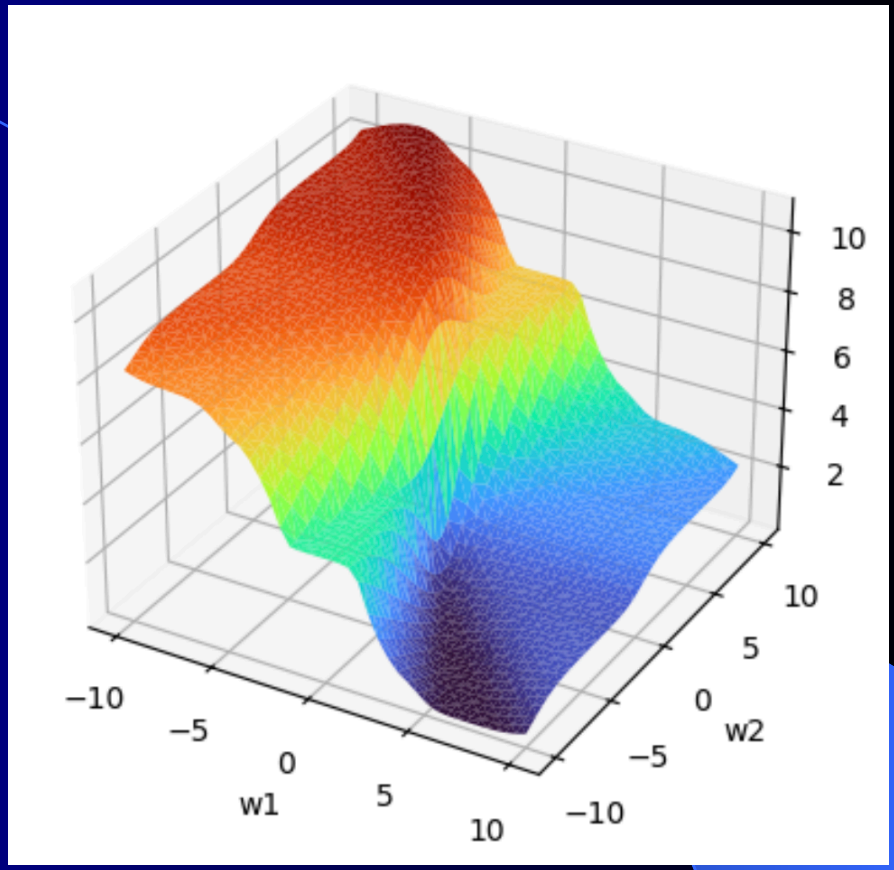


- Weights are updated even when the output would be correct
- Because this model is single layer and because of the SSE objective function, the error surface is guaranteed to be parabolic with only one minima
- Learning rate
  - If learning rate is too large can jump around global minimum
  - If too small, will get to minimum, but will take a longer time
  - Can decrease learning rate over time to give higher speed and still attain the global minimum (although exact minimum is still just for training set and thus…)

Perceptron Rule



Delta Rule + Sigmoid(net)



Changing to the Delta Rule and using Sigmoid(net) for output changes the decision surface to smooth and differentiable

Perceptron rule vs Delta rule

- Perceptron rule (target - thresholded output) guaranteed to converge to a separating hyperplane if the problem is linearly separable. Otherwise may not converge – could get in a cycle

- Singe layer Delta rule guaranteed to have only one global minimum. Thus, it will converge to the best SSE solution whether the problem is linearly separable or not.
  - Could have a higher misclassification rate than with the perceptron rule and a less intuitive decision surface – we will discuss this later with regression where Delta rules is more appropriate

- Stopping Criteria – For these models we stop when no longer making progress
  - When you have gone a few epochs with no significant improvement/ change between epochs (including oscillations)