Perceptrons

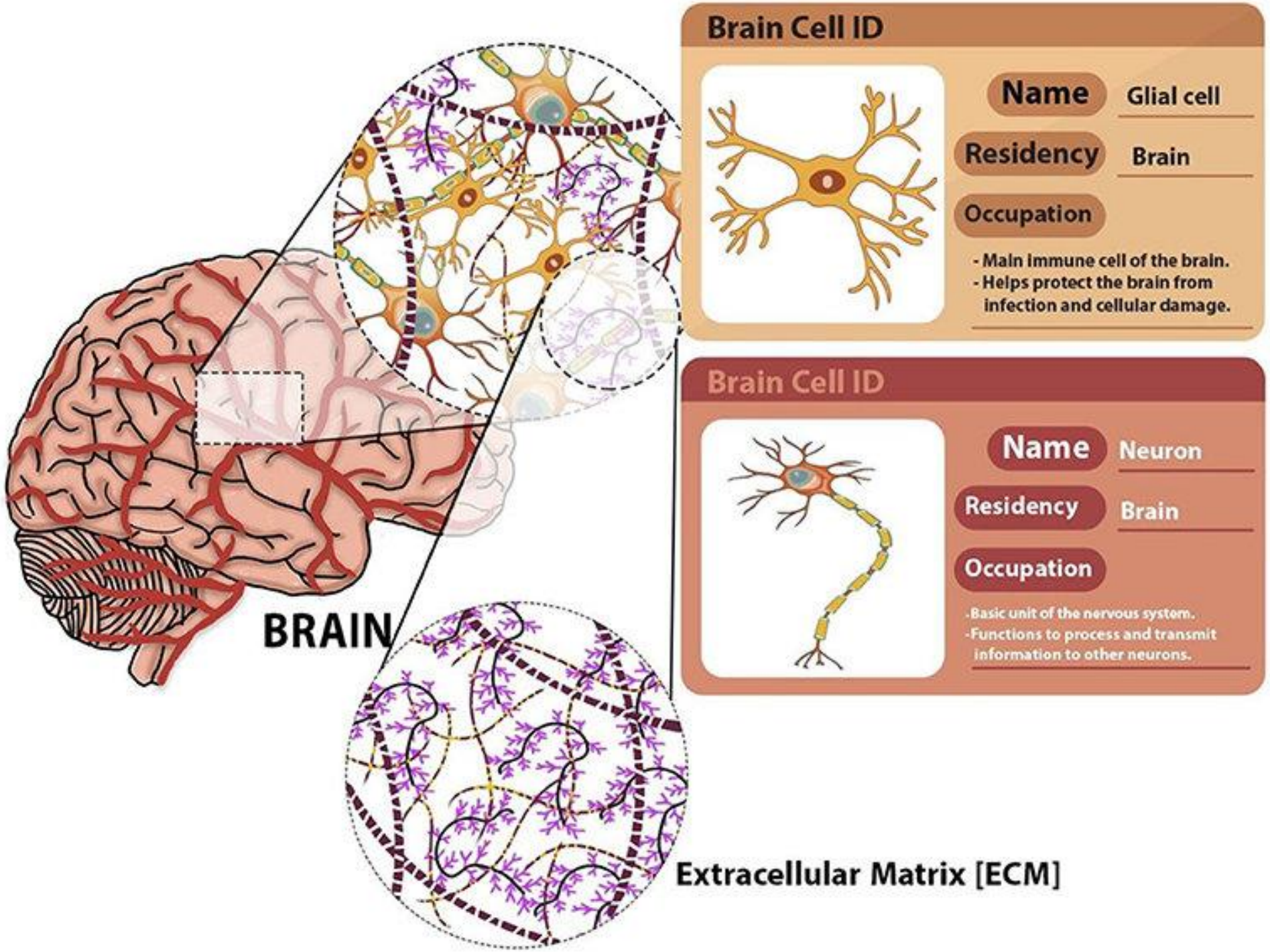
3 March 2026

Alex Lyman

Perceptron

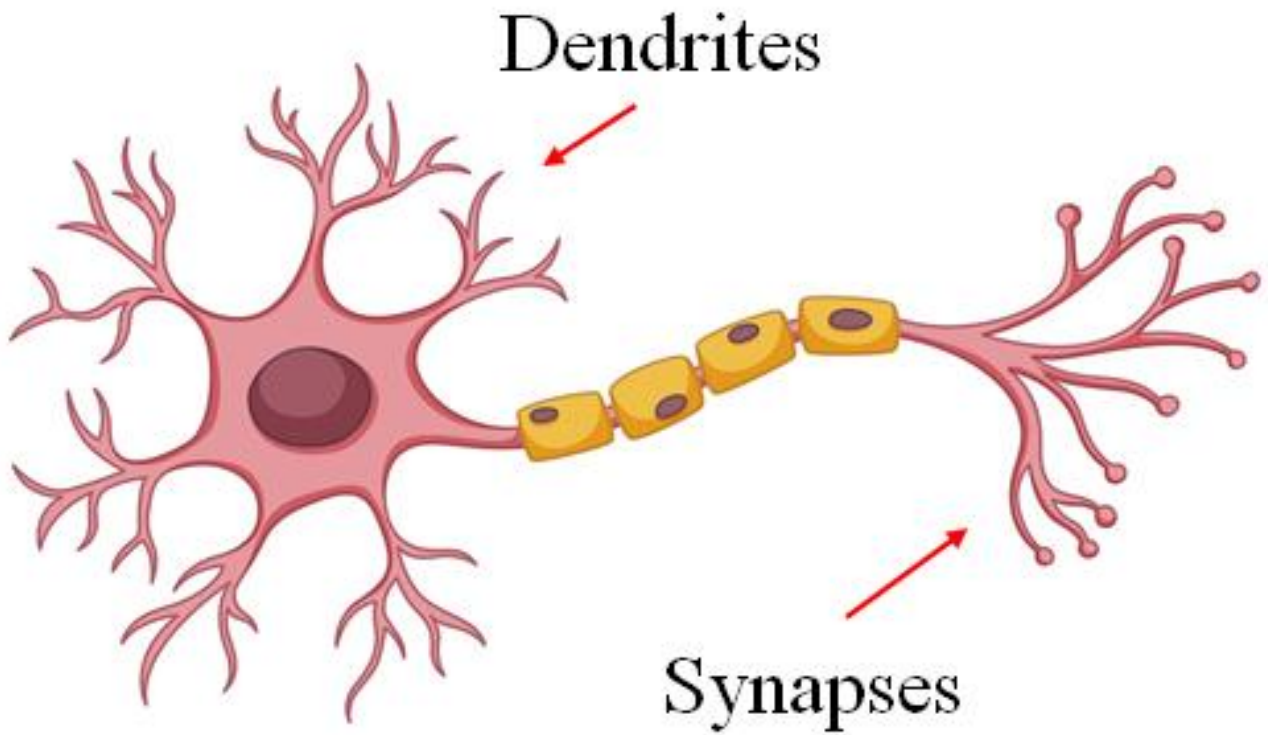
Neural Nets

- We've heard of neural nets before.
- What are they?
- Neural Nets – so named because their design looks kind of like neurons (brain cells)


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.
BRAIN
Extracellular Matrix [ECM]

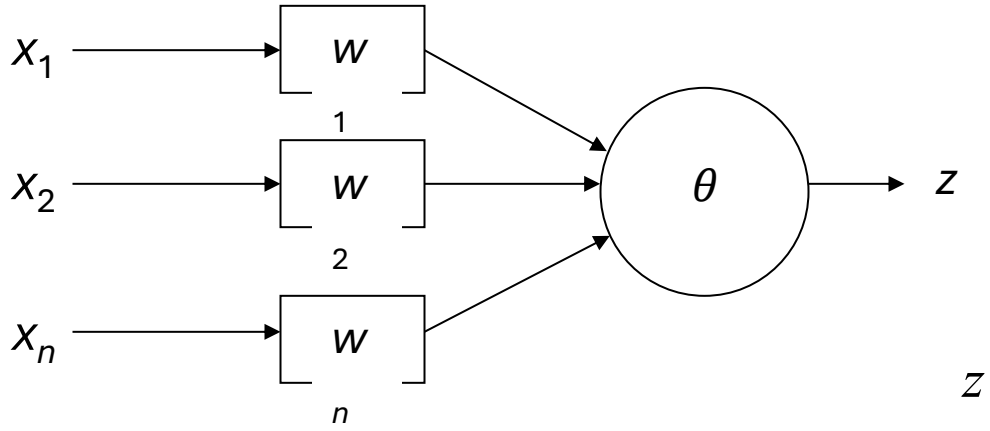
Neural Nets

- We've heard of neural nets before.
- What are they?
- Neural Nets – so named because their design looks kind of like neurons. (brain cells)
- First Neural Net was just one 'cell'. Called the perceptron.
- First neural network learning model in the 1960's
  - Frank Rosenblatt
- Simple and limited (single layer model)
- Concepts are the same in multi-layer neural nets.



NEURON

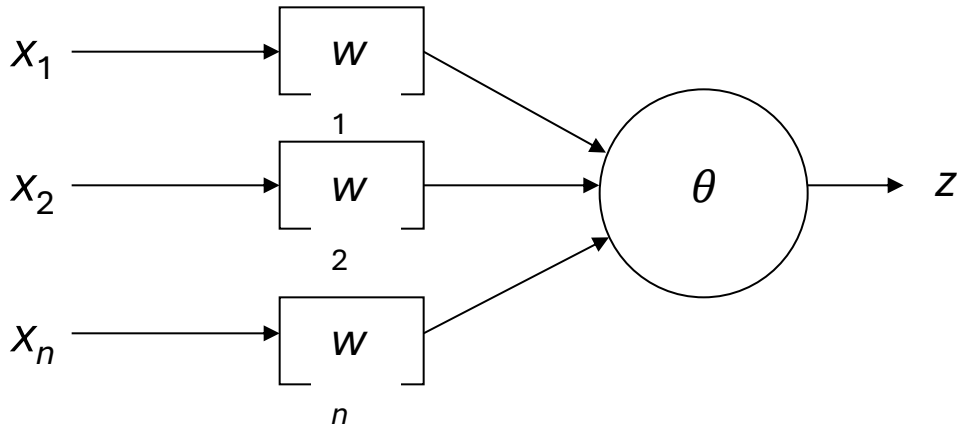
Perceptron Node – Threshold Logic Unit



$$z = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq \theta \\ 0 & \text{if } \sum_{i=1}^{n} x_i w_i < \theta \end{cases}$$

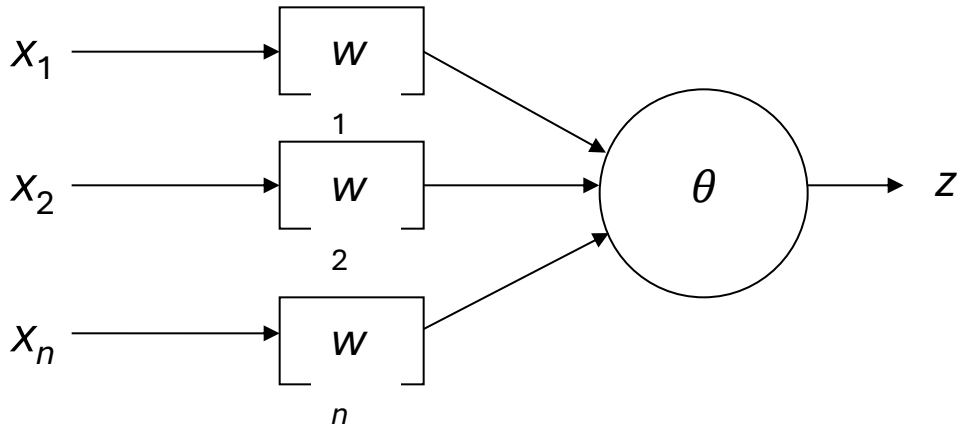
- x_1, x_2, x_n Inputs
- w_1, w_2, w_n Weights
- θ Threshold Function
- z Output

Perceptron Node – Threshold Logic Unit



- Learn weights such that an objective function is maximized.
- What objective function should we use?
- What learning algorithm should we use?

Perceptron Algorithm



- Initialize the weights and bias to zero or small random values.
- Iterate through the training data for multiple epochs.
  - Predict the output.
  - If a mistake occurs, update the weights and bias.
  - Repeat until all training examples are correctly classified, or a maximum number of iterations is reached.

Fake Model Example (review)

How Does a Model Learn?

- Imagine training a basketball-shooting cannon. You have 2 parameters (weights):
  - Angle (x)
  - Angle (y)

How Does a Model Learn?

- You start with the cannon pointing in some direction (randomly initialized weights or weights at zero).
- How do you know where the cannon will shoot?
- You have to try shooting the cannon.
- You miss. Now what?

How Does a Model Learn?

- You can see how far off you are.
- This is the error (related to loss)
- Then, you make a decision to try and minimize that error.
- In our case, we change our parameters (the x and y angles of the cannon) to try and make a better shot.

How Does a Model Learn?

- We repeat the process:
  - Shoot the cannon
  - Measure how far we missed by (error/loss)
  - Tweak the angle of the cannon (weights)

until we're making all of our shots

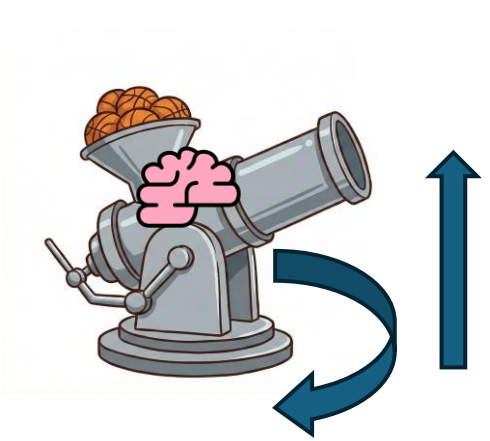
We can think of each time we shoot, measure, tweak as a training example.

How Does a Model Learn?

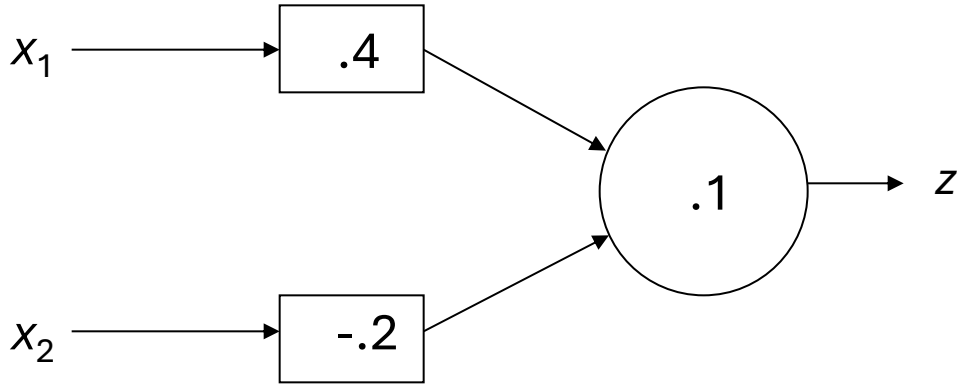
- But we don’t want to have to tweak the parameters by ourselves.
- Instead, we create some sort of system that tries to minimize error **by itself** by tweaking its own parameters.
- That’s the **machine** in **machine learning.**

How Does a Model Learn?

- So instead, we make a little mechanical brain for our cannon. (This is an abstraction). After each shot, we tell it its error, and allow it to tweak its own parameters.
- This is what the perceptron does!
- We've finally gotten to the 'little mechanical brain'



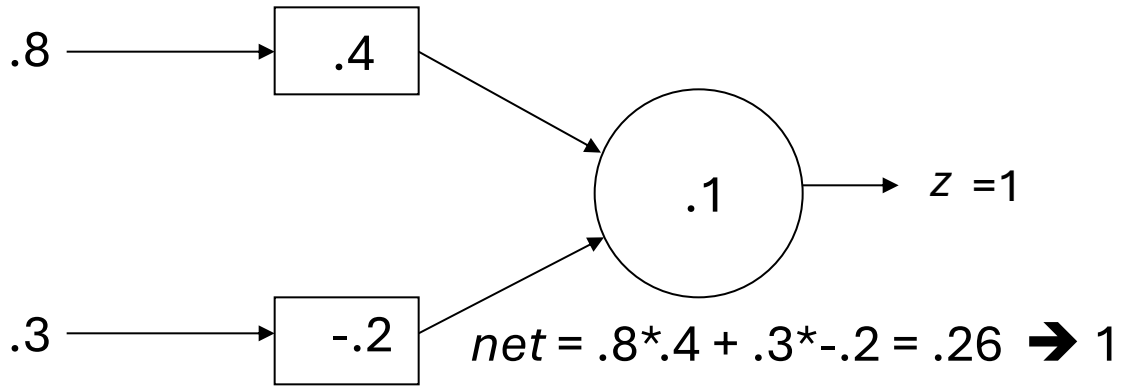
Perceptron Learning Algorithm



x_1	x_2	*target*
.8	.3	1
.4	.1	0

$$z = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq \theta \\ 0 & \text{if } \sum_{i=1}^{n} x_i w_i < \theta \end{cases}$$

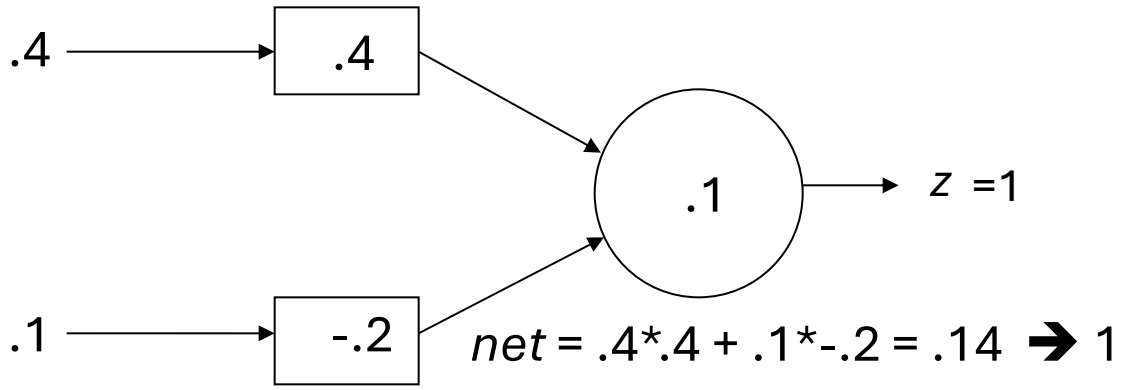
First Training Instance



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

$$z = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq \theta \\ 0 & \text{if } \sum_{i=1}^{n} x_i w_i < \theta \end{cases}$$

Second Training Instance



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

Need to make a correction
(tweak our weights)

$$z = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq \theta \\ 0 & \text{if } \sum_{i=1}^{n} x_i w_i < \theta \end{cases}$$

Perceptron Rule Learning

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

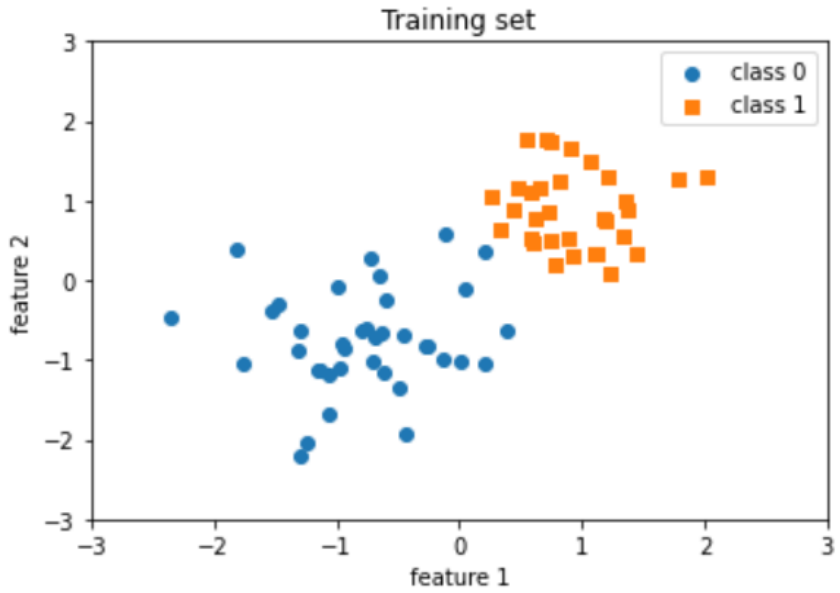
- Where w_i is the weight from input i to the perceptron node,
  - c is the learning rate, (hyperparameter)
  - t is the target for the current instance,
  - z is the current output,
  - $(t\text{-}z)$ is the error
  - x_i is i^{th} input

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

Perceptrons for Classification

- Perceptrons can be used to create a linear decision boundary by drawing a separating hyperplane.

$$z = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq \theta \\ 0 & \text{if } \sum_{i=1}^{n} x_i w_i < \theta \end{cases}$$



Note

- Perceptrons can take lots of different inputs. (More than two)
- However, 2-d graphs are easy to draw and think about.
- When we did linear regression, most of our hyperplanes were lines.
- However, when we trained models for linear regression homework, very few or none of them only had 2 features.
- Similarly, most perceptron math is hyperplane math, but we'll be doing lines in 2-d space so we can draw graphs.

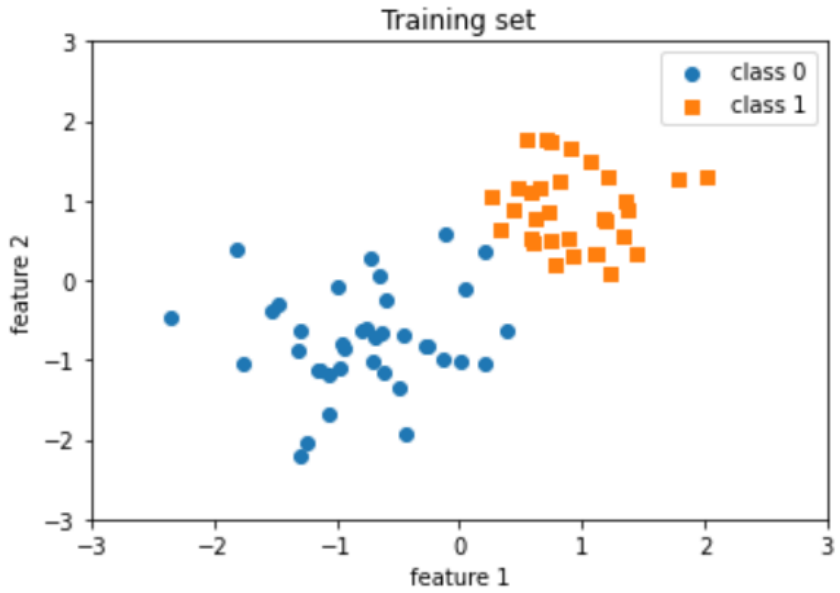
Perceptron Rule Learning

- Create a perceptron node with *n* inputs
- Iteratively select an example 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*
- *https://dictionary.cambridge.org/us/pronunciation/english/epoch*
- Continue training until total training set error ceases to improve
- Perceptron Convergence Theorem: Guaranteed to find a solution in finite time if a solution exists

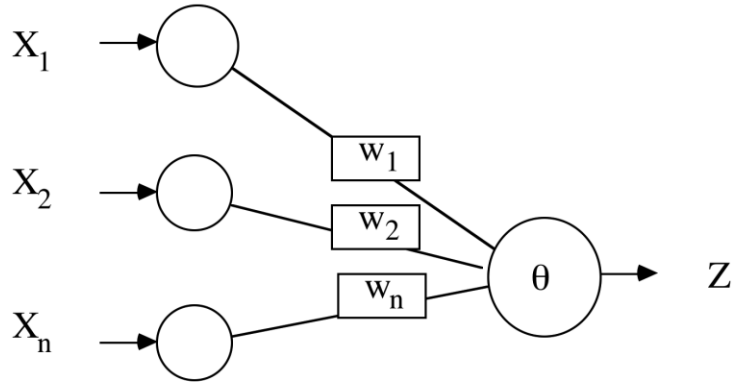
But Wait!

- The perceptrons we've looked at are missing a term:
  - $z=w_1x_1+w_2x_2$
  - $z=w_1x_1$
- To look like our equation from middle school:
  - $Y=mx+b$
  - $z=w_1x_1+b$



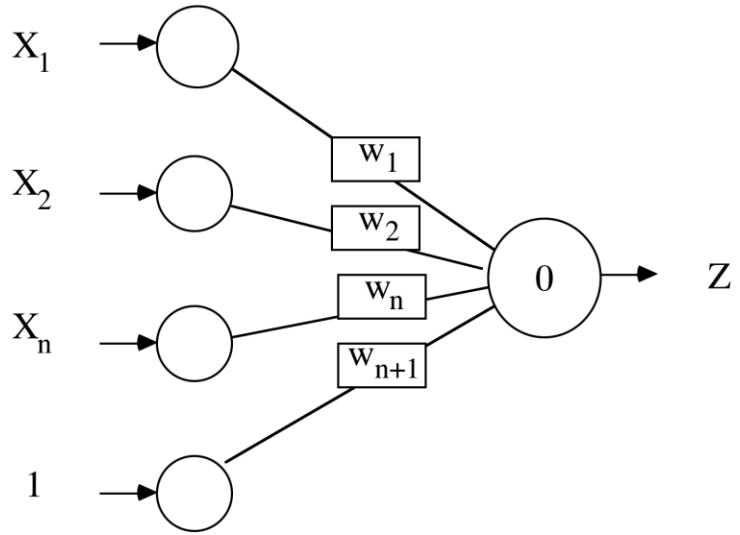
- What does that bias term do (in middle school math)?
  - (moves the line)

Weight Versus Threshold



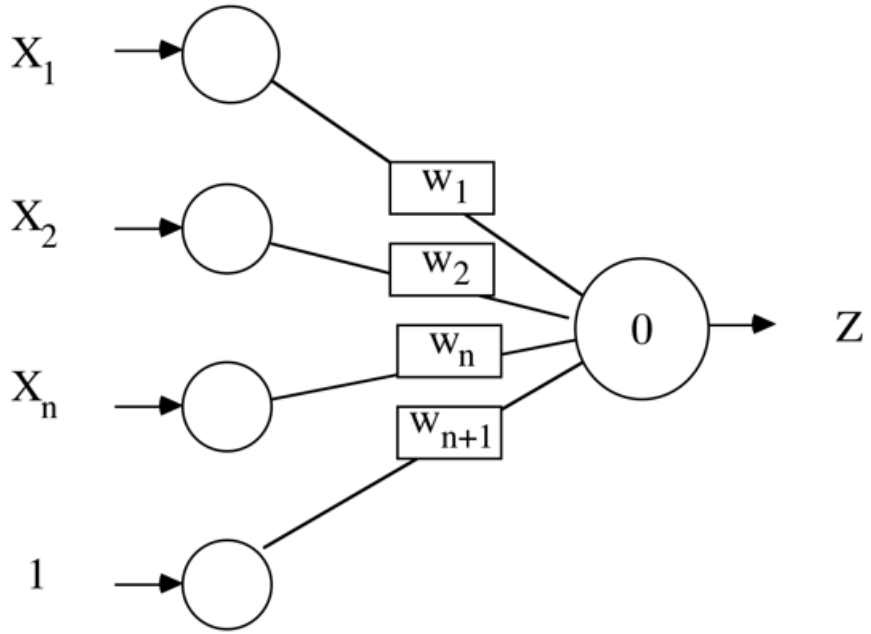
Do you need to adjust Theta? Yes, in most cases

We can learn the threshold by adding another node **with an input of 1** (bias)



where $w_{n+1} = -\theta$

Perceptron Algorithm



- Initialize the weights **and bias** to zero or small random values.
- Iterate through the training data for multiple epochs.
  - Predict the output.
  - If a mistake occurs, update the weights **and bias**.
  - Repeat until all training examples are correctly classified, or a maximum number of iterations is reached.

Numerical Example

Adding Bias to our Training Examples

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, we can just think of the bias as an extra available weight.
- Always use a bias weight

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

$$z = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq \theta \\ 0 & \text{if } \sum_{i=1}^{n} x_i w_i < \theta \end{cases}$$

Example	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

$$z = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq \theta \\ 0 & \text{if } \sum_{i=1}^{n} x_i w_i < \theta \end{cases}$$

Example	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

$$z = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq \theta \\ 0 & \text{if } \sum_{i=1}^{n} x_i w_i < \theta \end{cases}$$

Example	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			

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

$$z = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq \theta \\ 0 & \text{if } \sum_{i=1}^{n} x_i w_i < \theta \end{cases}$$

Example	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

$$z = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} x_i w_i \geq \theta \\ 0 & \text{if } \sum_{i=1}^{n} x_i w_i < \theta \end{cases}$$

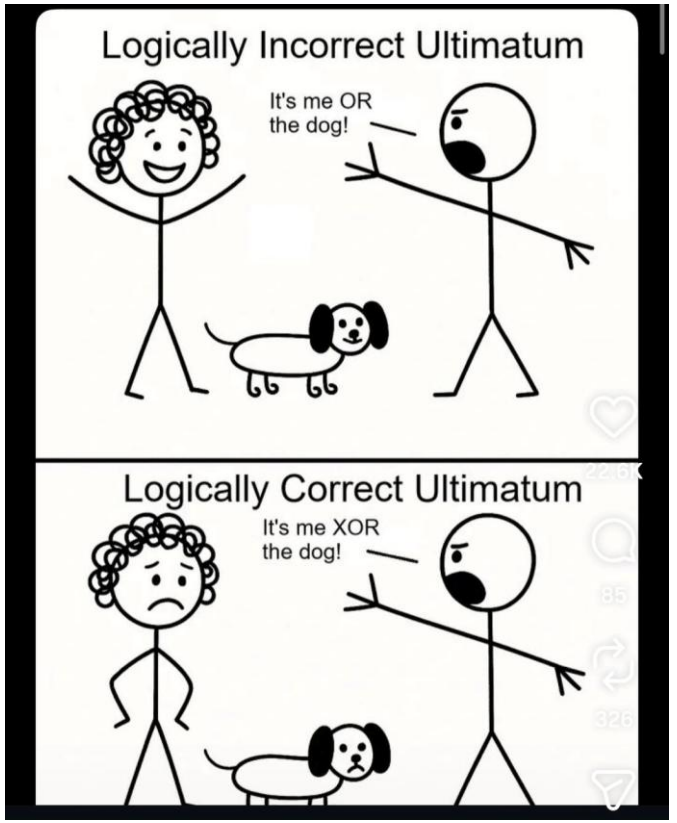
Example	Target (t)	Weight Vector (w_i)	Net	Output (z)	ΔW
0 0 1 1	0	0 0 0 0	0	0	0 0 0 0
1 1 1 1	1	0 0 0 0	0	0	1 1 1 1
1 0 1 1	1	1 1 1 1	3	1	0 0 0 0
0 1 1 1	0	1 1 1 1	3	1	0 -1 -1 -1
0 0 1 1	0	1 0 0 0			

Example

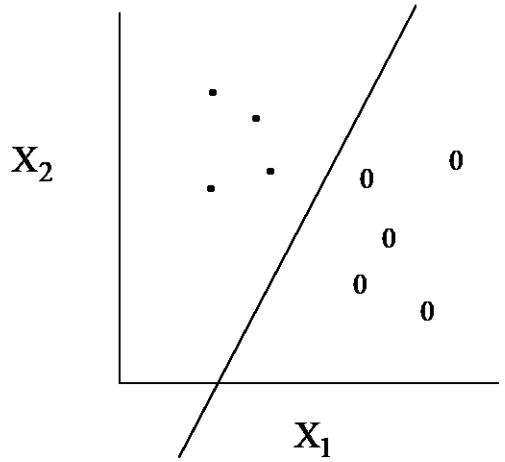
- Assume a 3 input perceptron plus bias (it outputs 1 if net > 0, else 0)
- Assume a learning rate c of 1 and initial weights all 0: $\Delta w_i = c(t - z)\, x_i$
- Training set
  0 0 1 -> 0
  1 1 1 -> 1
  1 0 1 -> 1
  0 1 1 -> 0

Example	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

Break Time!


Logically Incorrect Ultimatum
It's me OR the dog!
Logically Correct Ultimatum
It's me XOR the dog!

Linear Separability



2-d case (two inputs)

$$W_1X_1 + W_2X_2 > \theta \quad (Z=1)$$
$$W_1X_1 + W_2X_2 < \theta \quad (Z=0)$$

So, what is decision boundary?

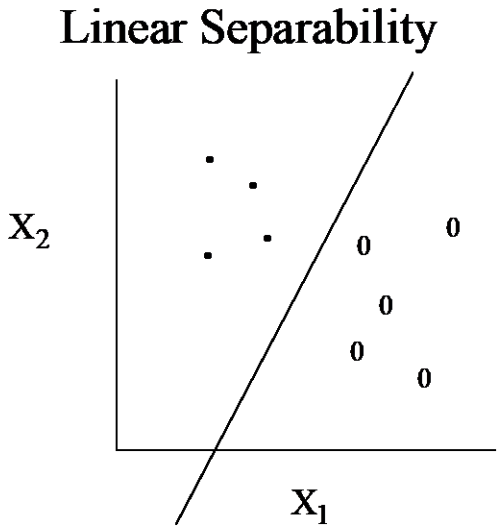
$$W_1X_1 + W_2X_2 = \theta$$
$$X_2 + W_1X_1/W_2 = \theta/W_2$$
$$X_2 = (-W_1/W_2)X_1 + \theta/W_2$$

$$Y = MX + B$$

If no bias weight, the hyperplane must go through the origin.
Note that since θ = -bias, the equation with bias is:
$X_2 = (-W_1/W_2)X_1 - bias/W_2$

M = $-W_1/W_2$
B = $-bias/W_2$

Note: bias is the weight for bias input



2-d case (two inputs)

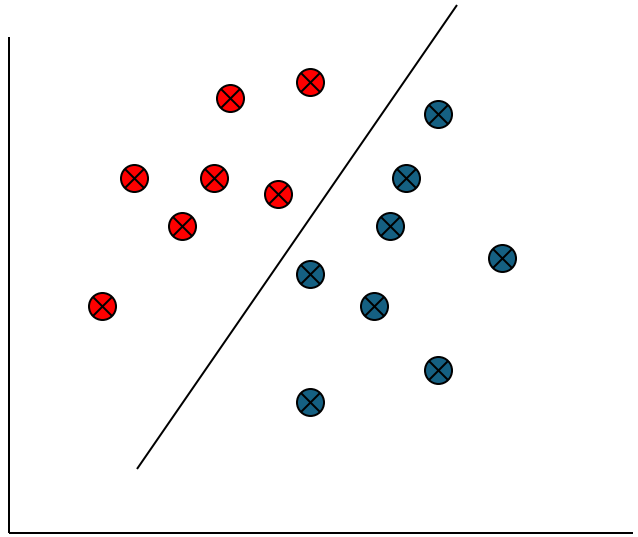
$$W_1X_1 + W_2X_2 > \theta \quad (Z=1)$$
$$W_1X_1 + W_2X_2 < \theta \quad (Z=0)$$

So, what is decision boundary?

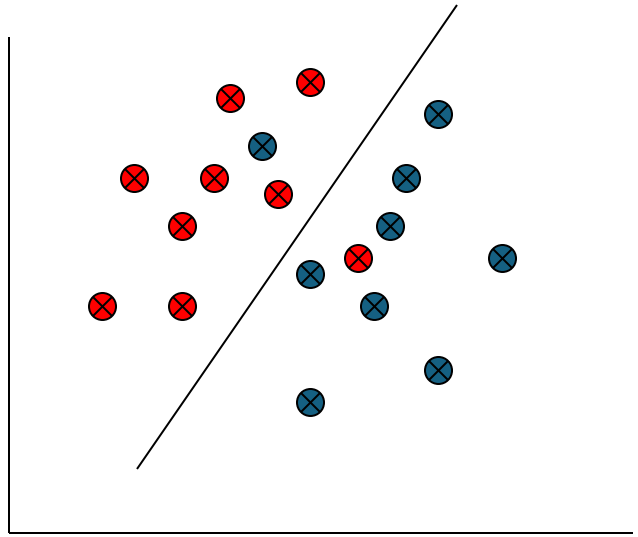
$$W_1X_1 + W_2X_2 = \theta$$
$$X_2 + W_1X_1/W_2 = \theta/W_2$$
$$X_2 = (-W_1/W_2)X_1 + \theta/W_2$$

$$Y = MX + B$$

Linear Separability

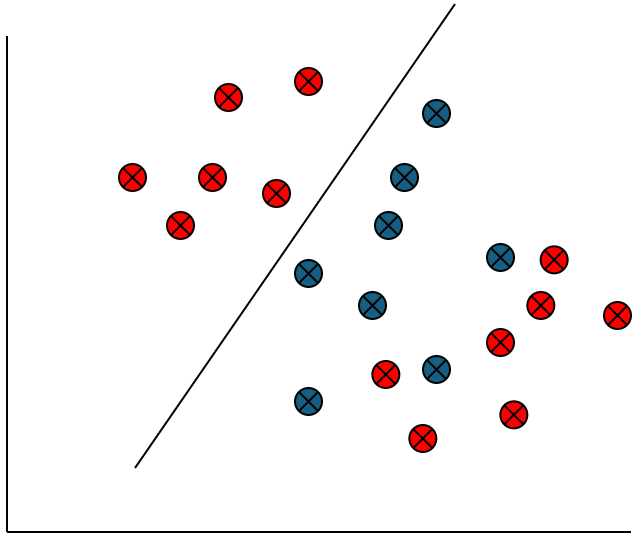


Linear Separability and Generalization



When is data noise vs. a legitimate exception

Limited Functionality of Hyperplane



Linear Models which are Non-Linear in the Input Space

- So far we have used $f(\mathbf{x}, \mathbf{w}) = sign(\sum_{1=1}^{n} w_i x_i)$
- We could preprocess the inputs in a non-linear way and do

$$f(\mathbf{x}, \mathbf{w}) = sign(\sum_{1=1}^{m} w_i \phi_i(\mathbf{x}))$$

- To the perceptron algorithm it is the same but with more/different inputs. It still uses the same learning algorithm.
- 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?

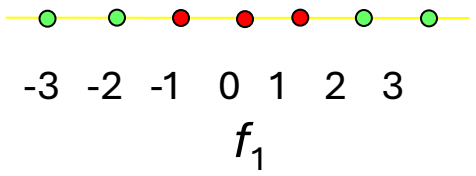
Quadric Machine

- All quadratic surfaces (2^{nd} order)
  - ellipsoid
  - parabola
  - etc.
- That significantly increases the number of problems that can be solved
- Can we solve XOR with this setup?

Quadric Machine

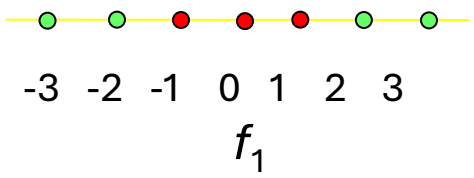
- All quadratic surfaces (2nd order)
  - ellipsoid
  - parabola
  - etc.
- That significantly increases the number of problems that can be solved
- But still many problems which are not quadrically separable
- Could go to 3rd and higher order features, but number of possible features grows exponentially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

Simple Quadric Example



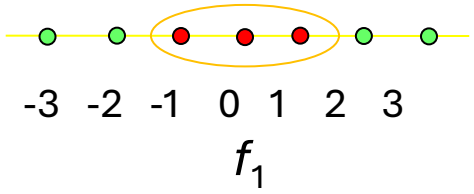
- What is the decision surface for a 1-d (1 input) problem?
- Perceptron with just feature f_1 cannot separate the data
- Could we add a transformed feature to our perceptron?

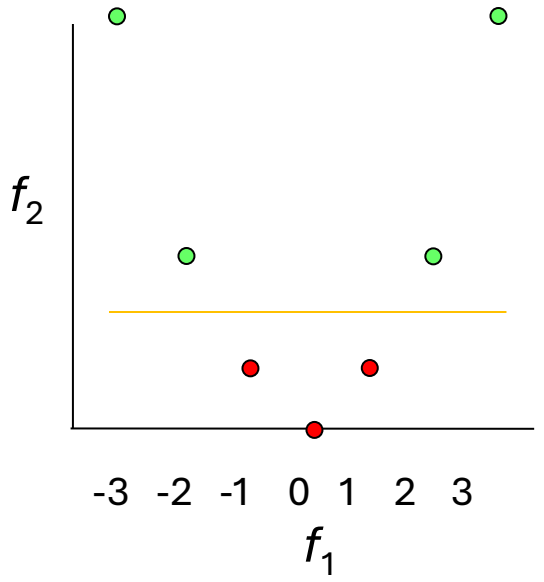
Simple Quadric Example



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

Simple Quadric Example





- Perceptron with just feature f_1 cannot separate the data
- Could we add another feature to our perceptron $f_2 = f_1^2$
- Note could also think of this as just using feature f_1 but now allowing a quadric surface to divide the data
  - Note that f_1 not actually needed in this case

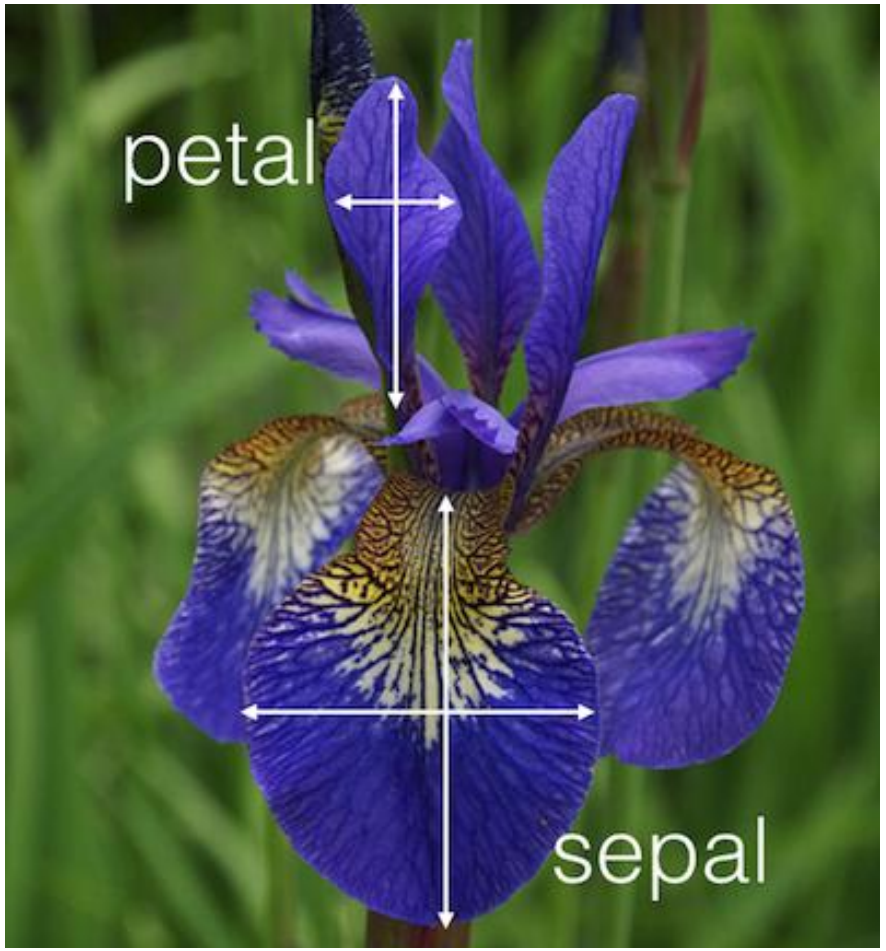
How to Handle Multi-Class Output

- 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
- Another approach: Create 1 perceptron for each pair of output classes, where the training set only contains examples from the 2 classes (one vs one)
  - Run all perceptrons on novel data and set the output to be the class with the most wins (votes) from the perceptrons
  - In case of a tie, use the net values to decide
  - Number of models grows by the square of the output classes

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



Quiz Time!

Code Time!