Doctrine and Covenants 88: 118

And as all have not faith, seek ye diligently
and teach one another words of wisdom; yea,
seek ye out of the best books words of wisdom;
seek learning, even by study and also by faith.

Ether 12:26

27 And if men come unto me I will show unto them their weakness. I give unto men weakness that they may be humble; and my grace is sufficient for all men that humble themselves before me; for if they humble themselves before me, and have faith in me, then will I make weak things become strong unto them.

Some things

- Homework:
  - Due before class on the day indicated
  - Turn in pdf or jpeg – make sure it is legible and easy to read
- Labs
  - Due at midnight
  - Turn in the .ipynb
- Look at a lab

Doing Your Labs

- Will use scikit-learn in individual labs
  - Whatever you want in group project
- Program in Python in Jupyter notebooks
  - NumPy library – Great with arrays, etc.
- Recommended tools and libraries
  - Colab – Google IDE for Python and Jupyter notebooks
  - Pandas – Data Frames and tools are very convenient
  - MatplotLib
  - Scikit-Learn

scikit-learn

- One of the most used and powerful machine learning toolkits out there
- Lots of implemented models and tools to use for machine learning applications
  - Sometimes missing some things we would like, but it is continually evolving
  - Source is available, and you can always override methods with your own, etc.
- Basically a Python Library to call from your Python code
- Familiarize yourself with the scikit-learn website as you will be using it for all labs

Machine Learning Tools

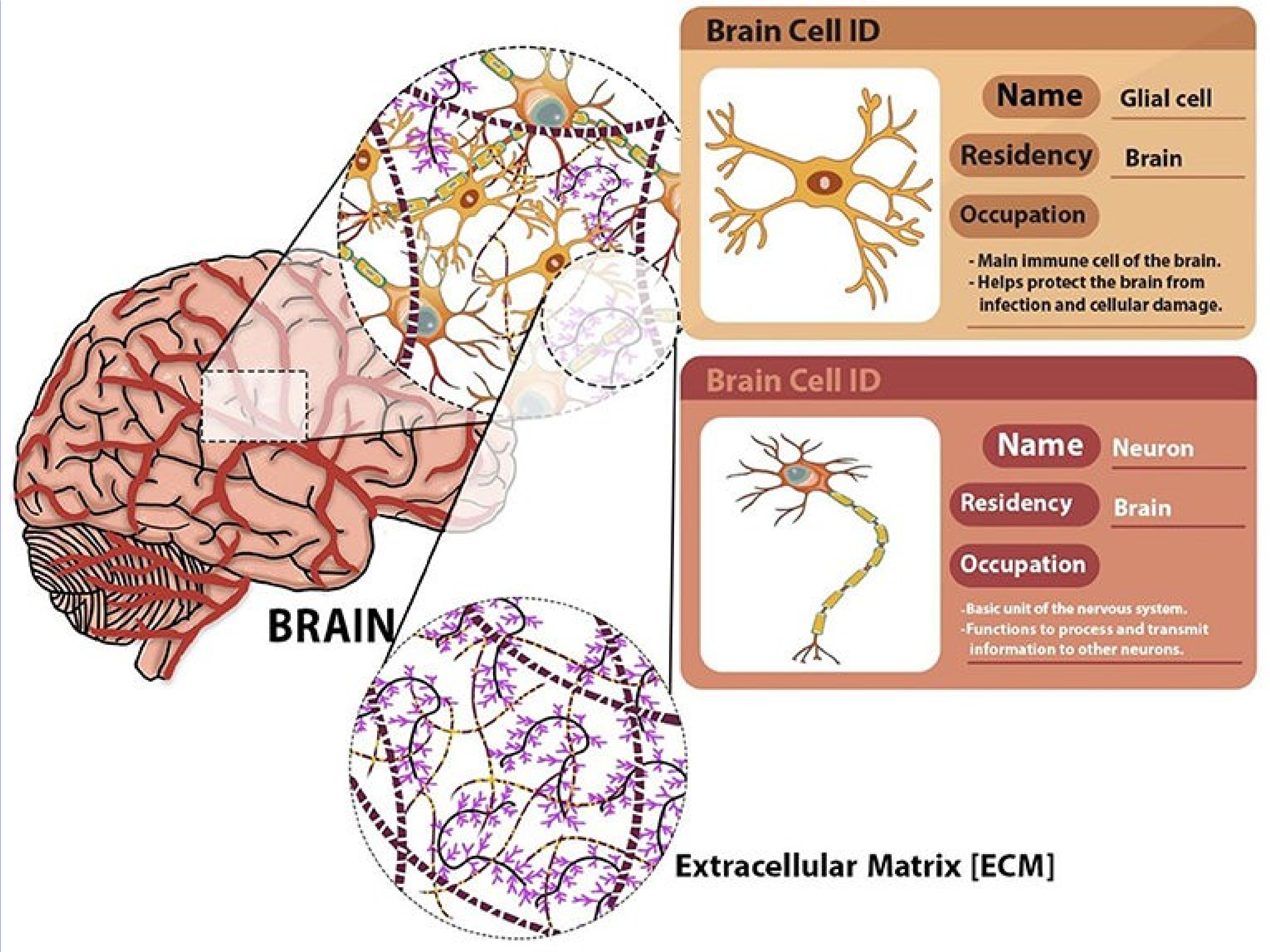
- Lots of new Machine Learning Tools
  - Weka was the first main site with lots of ready to run models
  - Scikit-learn now very popular
  - Languages:
    - Python with NumPy, matplotlib, pandas, other libraries
    - R (good statistical packages), but with growing Python libraries…
  - Deep Learning Neural Network frameworks – GPU capabilities
    - Tensorflow - Google
    - PyTorch – Multiple developers (Facebook, twitter, Nvidia...) - Python
    - Others: Caffe2 (Facebook), Keras, Theano, CNTK (Microsoft)
  - Data Mining Business packages – Visualization, Expensive
- Great for experimenting and applying to real problems
- But important to “get under the hood” and not just be black box ML users

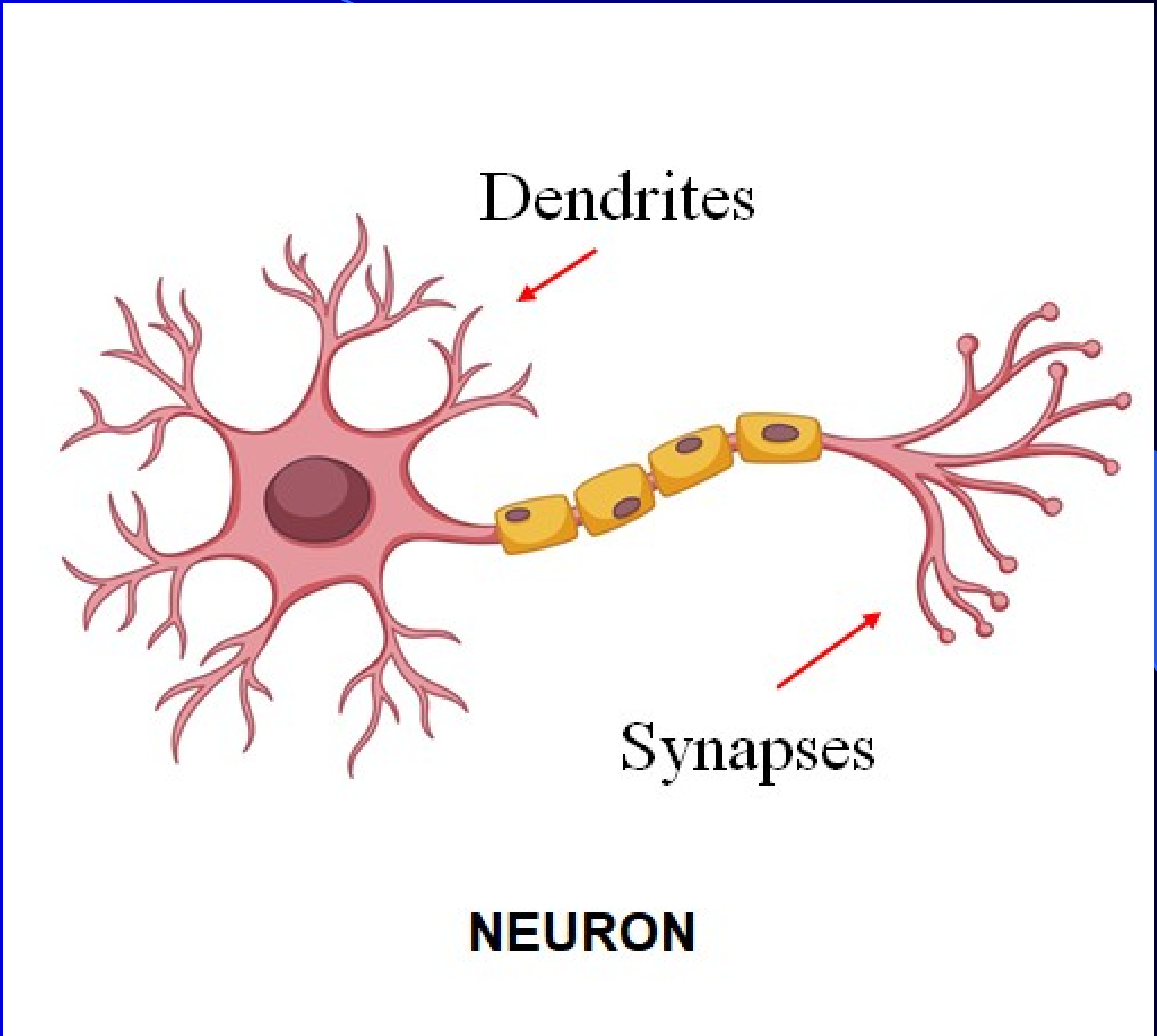
Perceptron Project

- Content Section of LS (Learning Suite) for project specifications
  - Review carefully the introductory part regarding all projects
- For each project carefully read the specifications for the lab in the Jupyter notebook on GitHub
- You can just copy the Perceptron notebook from the GitHub site to your computer and then add your work in the code and text boxes

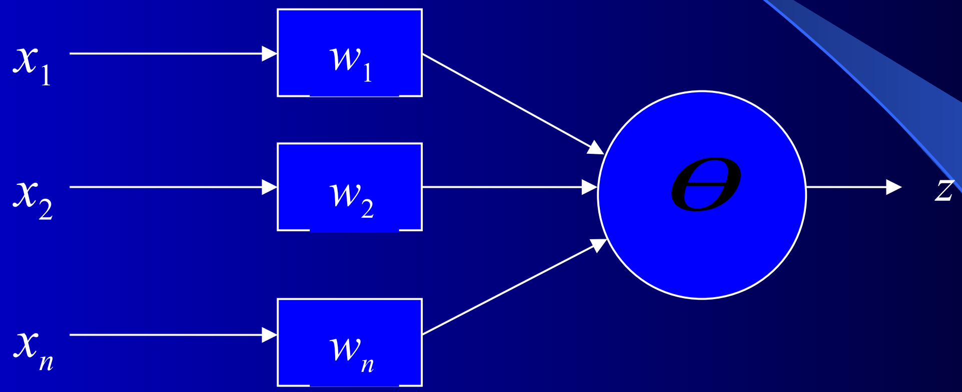
Perceptron Learning Algorithm

- 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.)


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]

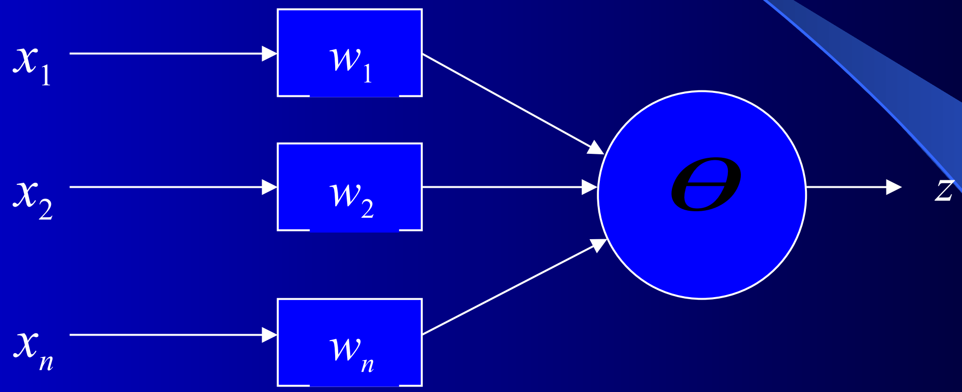

Dendrites
Synapses
NEURON

Perceptron Node – Threshold Logic Unit



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

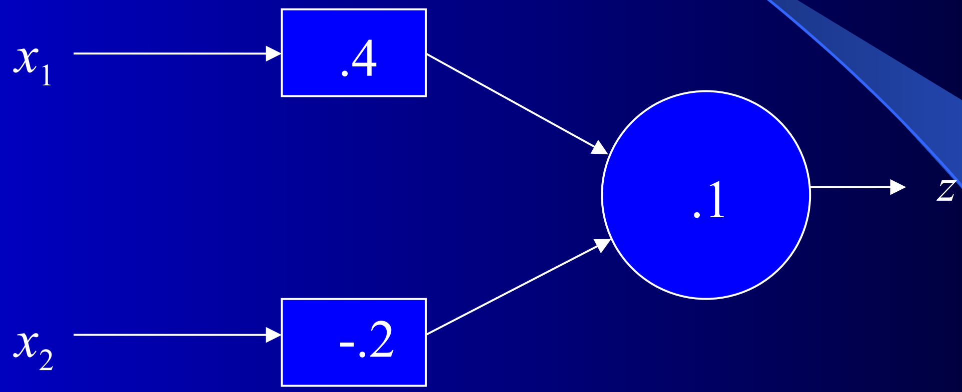
Perceptron Node – Threshold Logic Unit



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

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

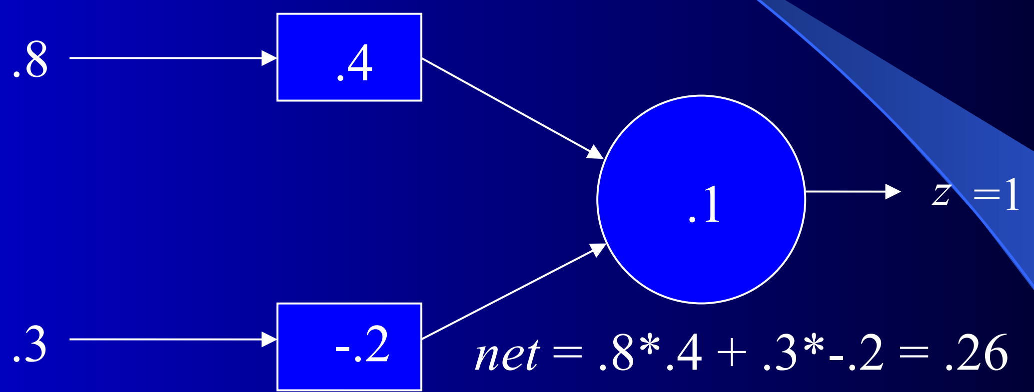
Perceptron Learning Algorithm



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

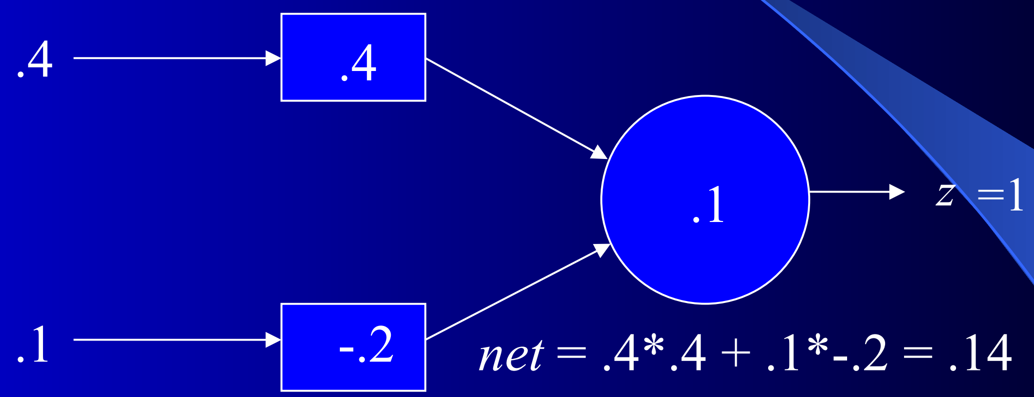


$$net = .8*.4 + .3*-.2 = .26 \rightarrow 1$$

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



$net = .4*.4 + .1*-.2 = .14 \rightarrow 1$

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

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

Need to make a correction

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,
  - 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 pattern correct
  - Scale by input value x_i

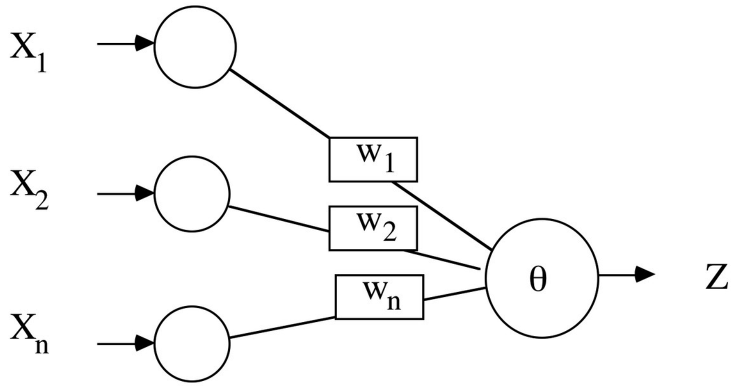
$$w_{ij} \leftarrow w_{ij} - \eta(y_j - t_j) \cdot x_i$$

Perceptron Rule Learning

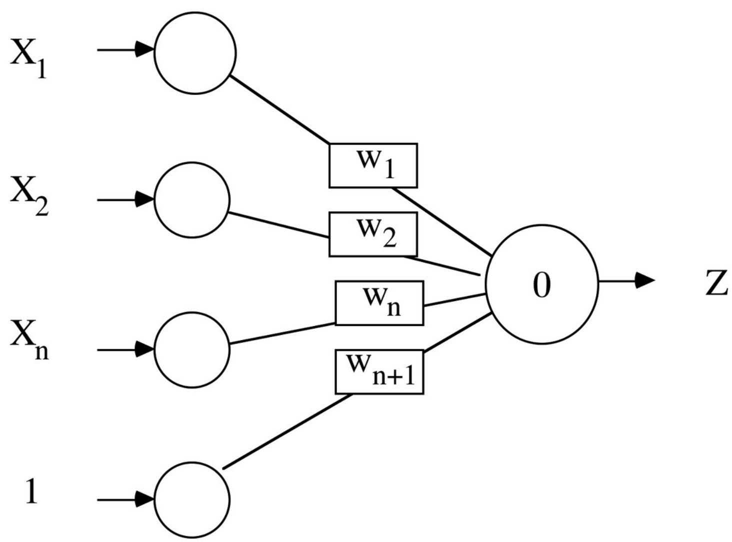
- Create a perceptron node with *n* inputs
- Iteratively select a pattern from the training set
- Calculate the output value z
- Apply the perceptron rule to adjust weights

- Each iteration through the training set is an *epoch*
- Continue training until total training set error ceases to improve
- Perceptron Convergence Theorem: Guaranteed to find a solution in finite time if a solution exists

Weight Versus Threshold



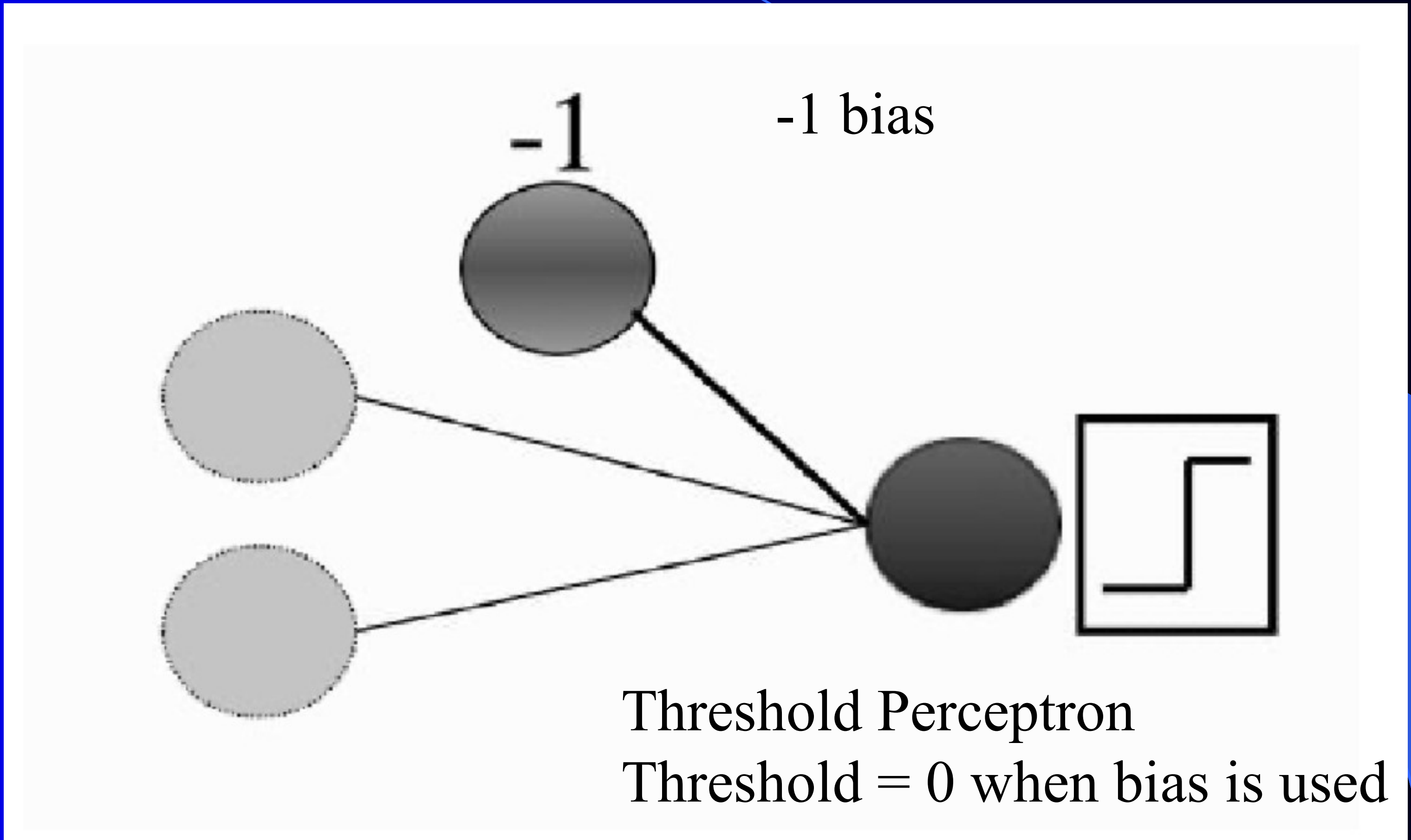
Do you need to adjust Theta? Yes, in most cases



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

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

From the textbook



Augmented Pattern Vectors

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 $-\theta$ 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 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

Pattern	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

Pattern	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

Pattern	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

Pattern	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

slido

Once it converges, the final weight vector will be

ⓘ Start presenting to display the poll results on this slide.

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

Pattern	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

Pattern	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

Pattern	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

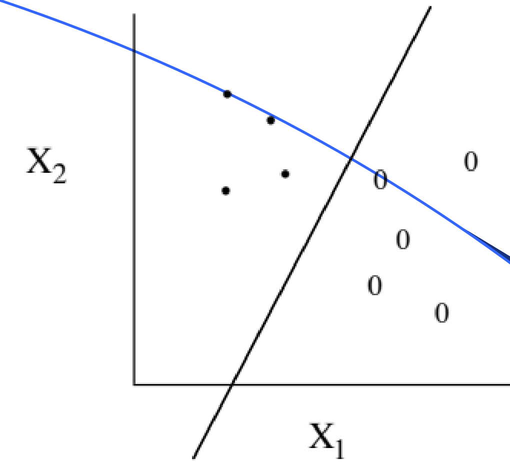
Perceptron Homework

- 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 pattern for just one epoch
- Training set 1 0 1 -> 0
  1 .5 0 -> 0
  1 -.4 1 -> 1
  0 1 .5 -> 1

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

Code

Linear Separability



2-d case (two inputs)

$$W_1X_1 + W_2X_2 > \theta \;\; (Z=1)$$

$$W_1X_1 + W_2X_2 < \theta \;\; (Z=0)$$

So, what is decision boundary?

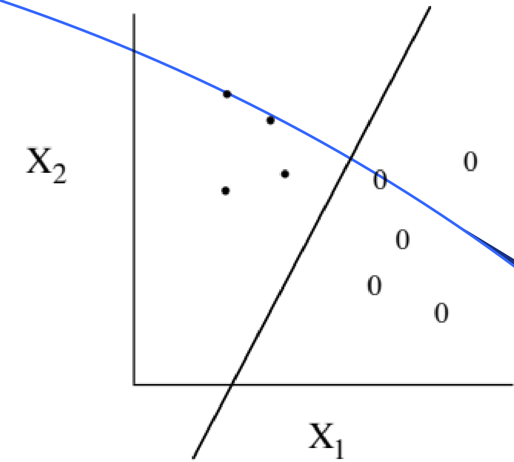
$$W_1X_1 + W_2X_2 = \theta$$

$$X_2 + W_1X_1/W_2 = \theta/W_2$$

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

$$Y = MX + B$$

Linear Separability



2-d case (two inputs)

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

So, what is decision boundary?

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

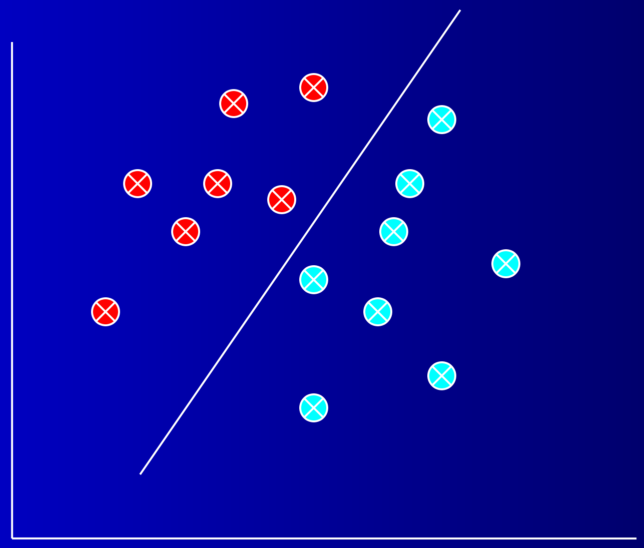
$$Y = MX + B$$

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

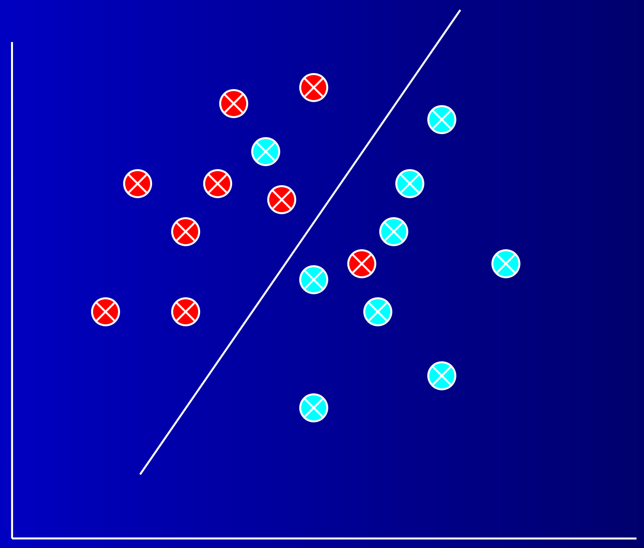
$M = -W_1/W_2$
$B = -bias/W_2$

Note: bias is the weight for bias input

Linear Separability

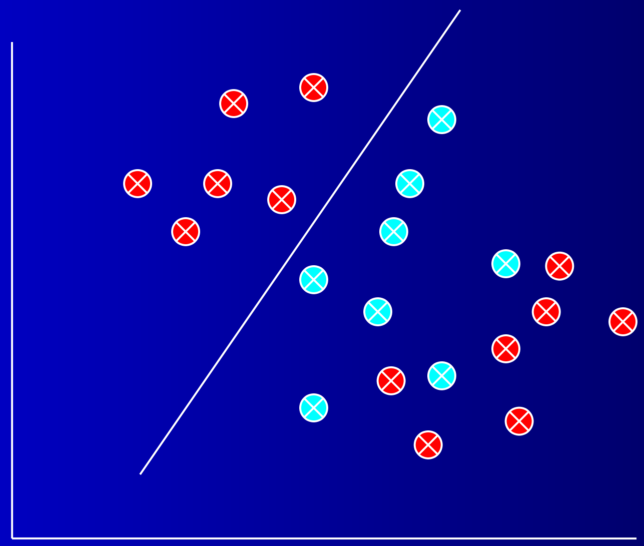


Linear Separability and Generalization

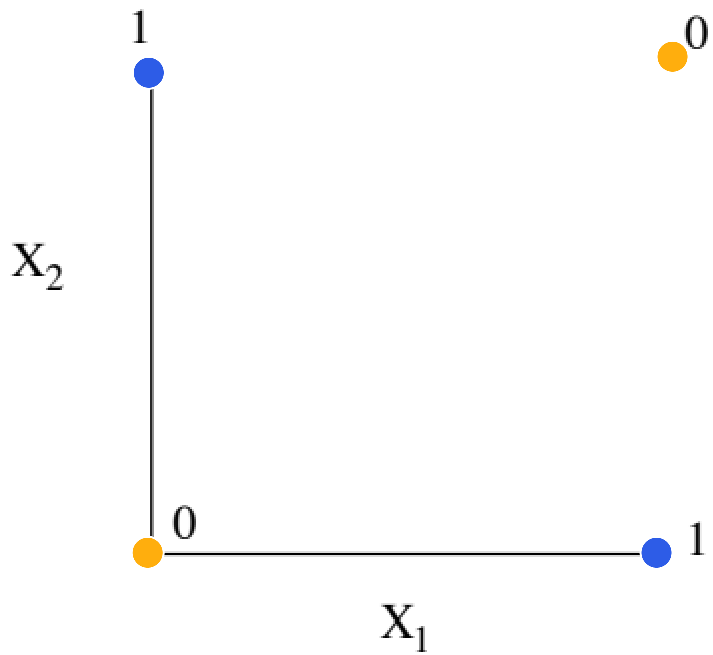


When is data noise vs. a legitimate exception

Limited Functionality of Hyperplane



Exclusive Or



Is there a dividing hyperplane?

Linear Models which are Non-Linear in the Input Space

- So far we have used $$f(\mathbf{x},\mathbf{w}) = sign(\sum_{i=1}^{n} w_i x_i)$$
- We could preprocess the inputs in a non-linear way and do $$f(\mathbf{x},\mathbf{w}) = sign(\sum_{i=1}^{m} w_i \phi_i(\mathbf{x}))$$
- 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 $\boldsymbol{x^2}$, $\boldsymbol{y^2}$, **and** $\boldsymbol{x \cdot y}$
- 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?

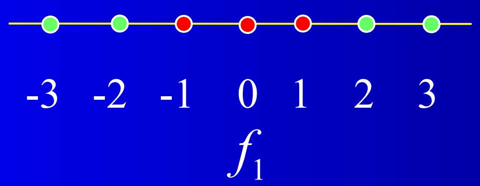
Quadric Machine

- 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

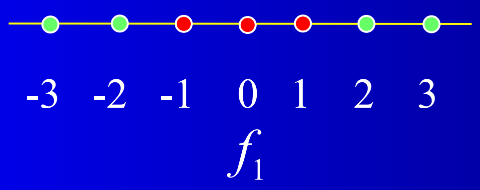
- All quadratic surfaces (2^{nd} order)
  - ellipsoid
  - parabola
  - etc.
- That significantly increases the number of problems that can be solved
- But still many problems which are not quadrically separable
- Could go to 3^{rd} and higher order features, but number of possible features grows exponentially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

Simple Quadric Example



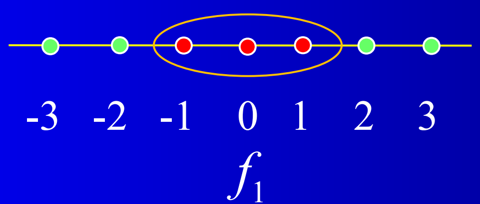
- What is the decision surface for a 1-d (1 input) problem?
- Perceptron with just feature f_1 cannot separate the data
- Could we add a transformed feature to our perceptron?

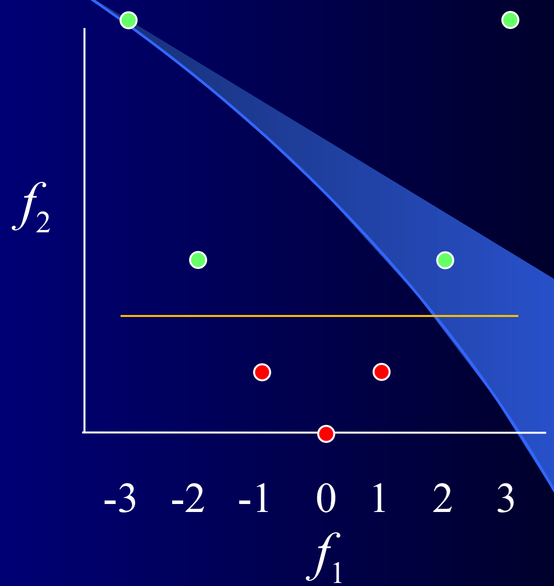
Simple Quadric Example



- 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

Quadric Machine Homework

- Assume a 2-input perceptron expanded to be a quadric (2^{nd} 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 pattern for one epoch with the following training set

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

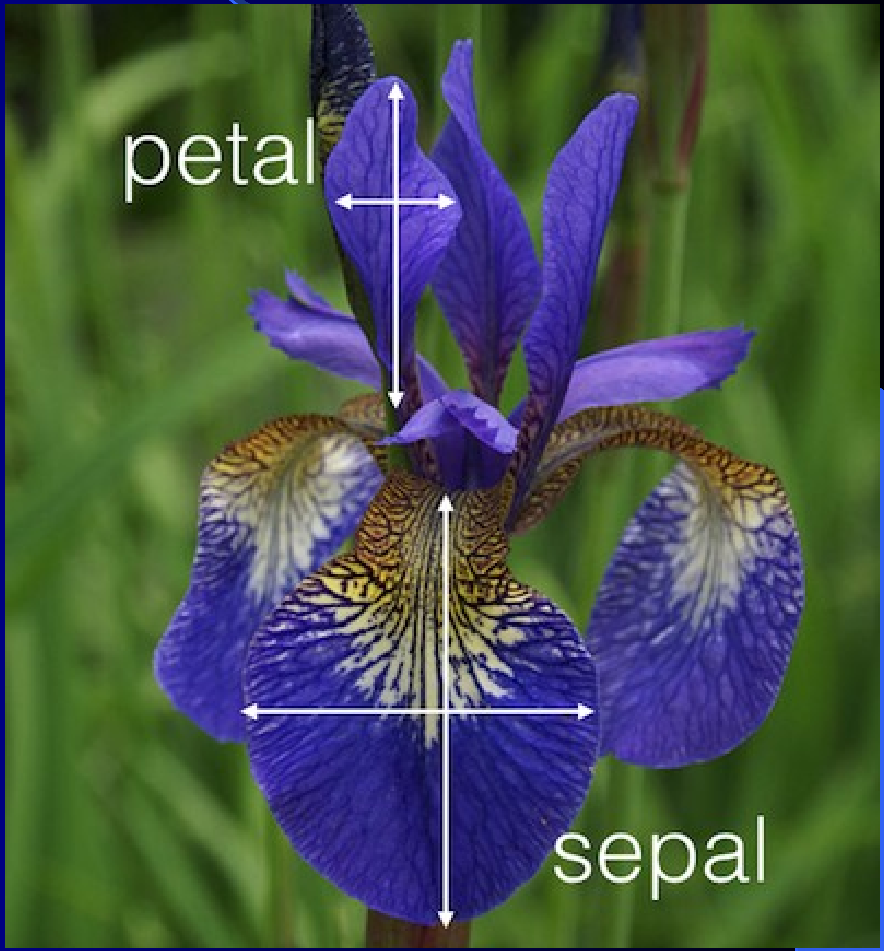
How to Handle Multi-Class Output

- 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
- 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

- Password is - perceptron

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
  - Pattern error = Target – output: Errors could cancel each other
    - $\Sigma|t_j - z_j|$ (L1 loss), where j indexes all outputs in the pattern
    - $\Sigma(t_j - z_j)^2$ (L2 loss), sum squared error (SSE)
  - L2 loss is generally preferred except in cases of large outliers

- For nominal data, pattern 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 pattern
- 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 root of the total data set MSE, and NOT the sum of the root of each individual pattern MSE

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 .67 .82
E. None of the above

slido

What are the errors for the data?

ⓘ Start presenting to display the poll results on this slide.

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				1.6
SSE				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 .67 .82
E. None of the above

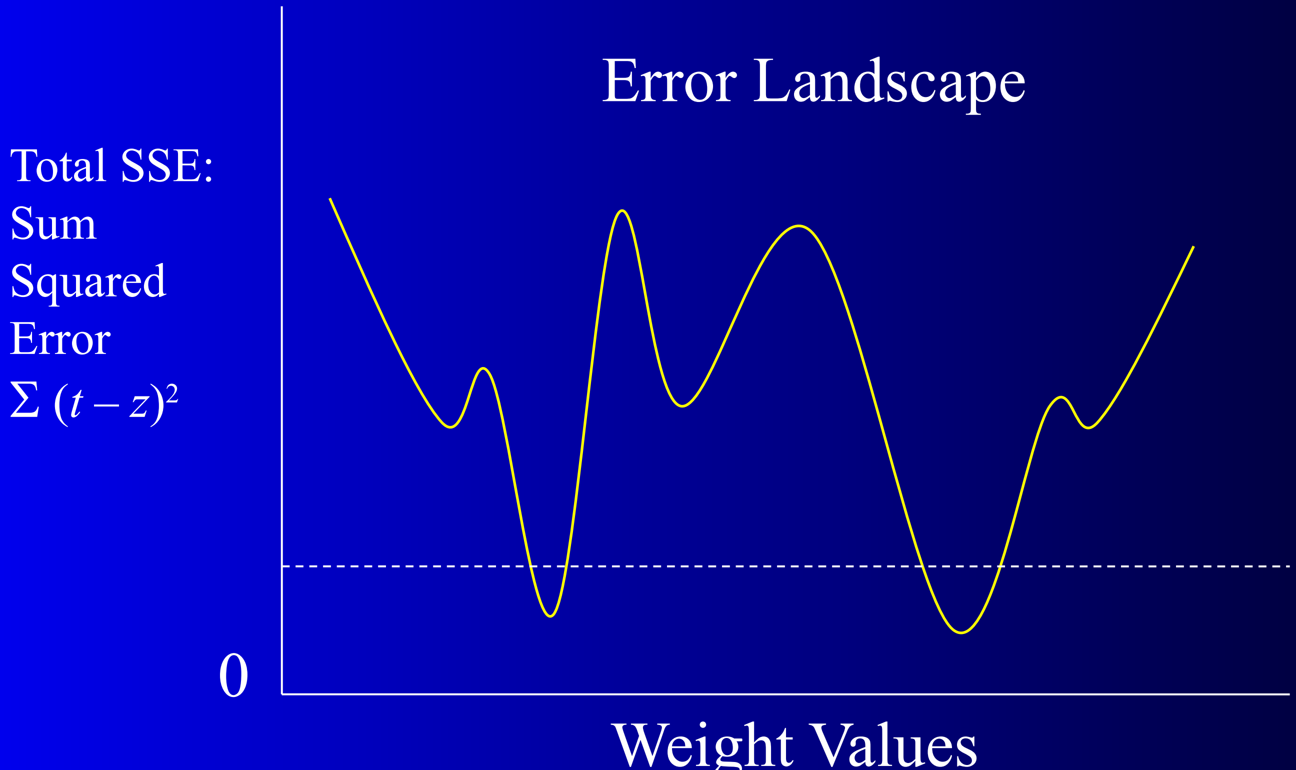
Error Values Homework

- Given the following data set, what is the L1, SSE (L2), MSE, and RMSE error of Output1, Output2, and the entire data set? Fill in cells that have a ?.
  - Notes: For instance 1 the L1 pattern error is 1 + .4 = 1.4 and the SSE pattern error is 1 + .16 = 1.16. The Data Set L1 and SSE errors will just be the sum of each of the pattern errors.

Instance	x	y	Output1	Target1	Output2	Target 2	Data Set
1	-1	-1	0	1	.6	1.0	
2	-1	1	1	1	-.3	0	
3	1	-1	1	0	1.2	.5	
4	1	1	0	0	0	-.2	
L1			?		?		?
SSE			?		?		?
MSE			?		?		?
RMSE			?		?		?

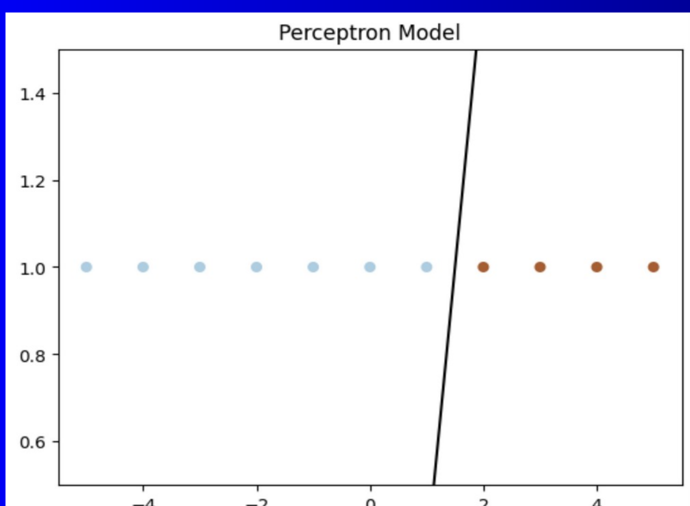
Error Surface

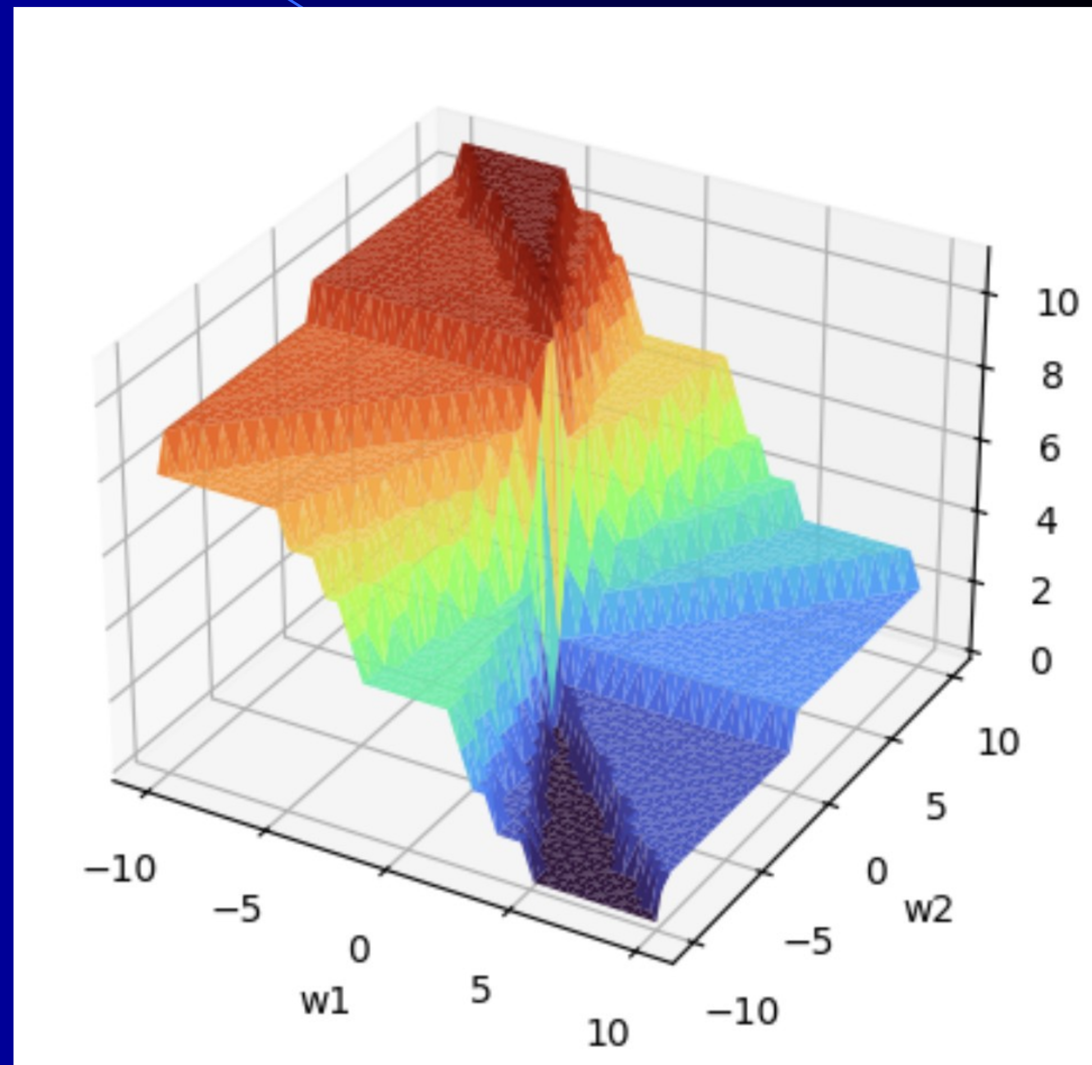
- Error is a function of the weights
  - $E = \Sigma(t_i - z_i)^2 = \Sigma(t_i - \Sigma x_j w_{ij})^2$
- If we could search this space, we could find the minimum



Error Surface

- Single perceptron – 1d input with bias





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

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

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

GRADIENT DESCENT VISUALIZATION

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 $E = \Sigma\, (t - z)^2$
- 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

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i}\frac{1}{2}\sum_{d\in D}(t_d - o_d)^2$$

$$= \frac{1}{2}\sum_{d\in D}\frac{\partial}{\partial w_i}(t_d - o_d)^2$$

$$\boxed{\frac{\partial E}{\partial w_i} = \sum_{d\in D}(t_d - o_d)(-x_{id})}$$

$$= \frac{1}{2}\sum_{d\in D} 2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - o_d)$$

Chain rule

$$= \frac{1}{2}\sum_{d\in D} 2(t_d - o_d)\frac{\partial}{\partial w_i}(t_d - \vec{w}.\vec{x})$$

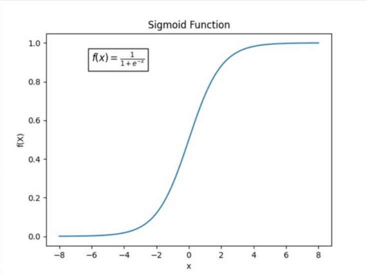
$$\boxed{\Delta w_i = \eta\sum_{d\in D}(t_d - o_d)x_{id}}$$

Delta rule algorithm

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

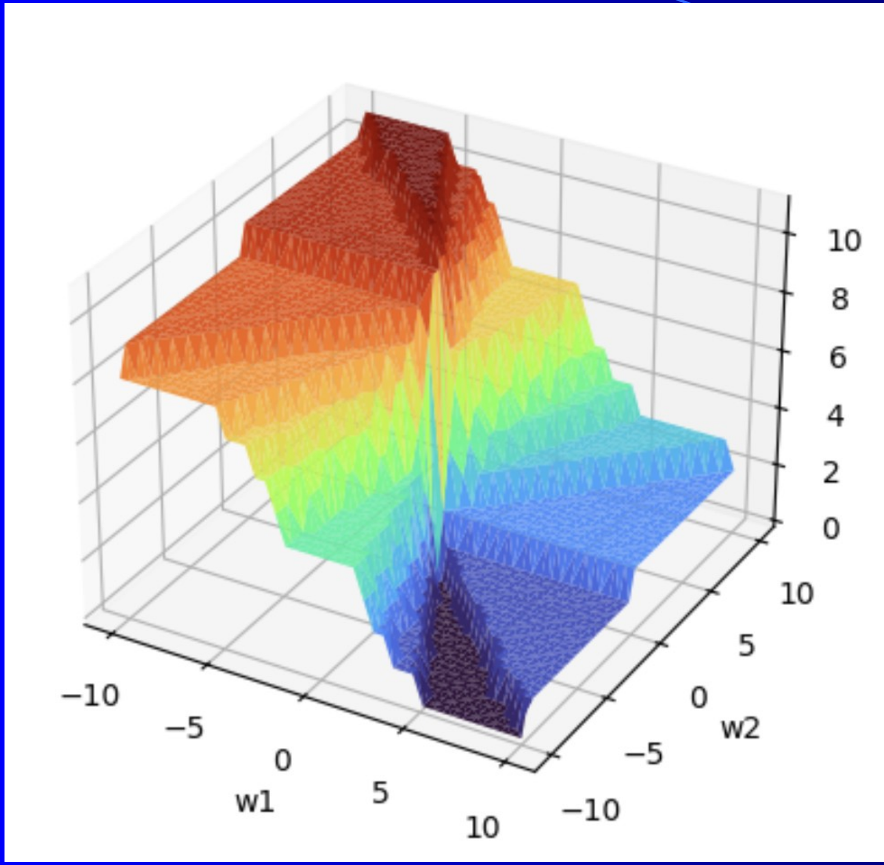
$$\Delta w_i = c(t - net)x_i$$

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

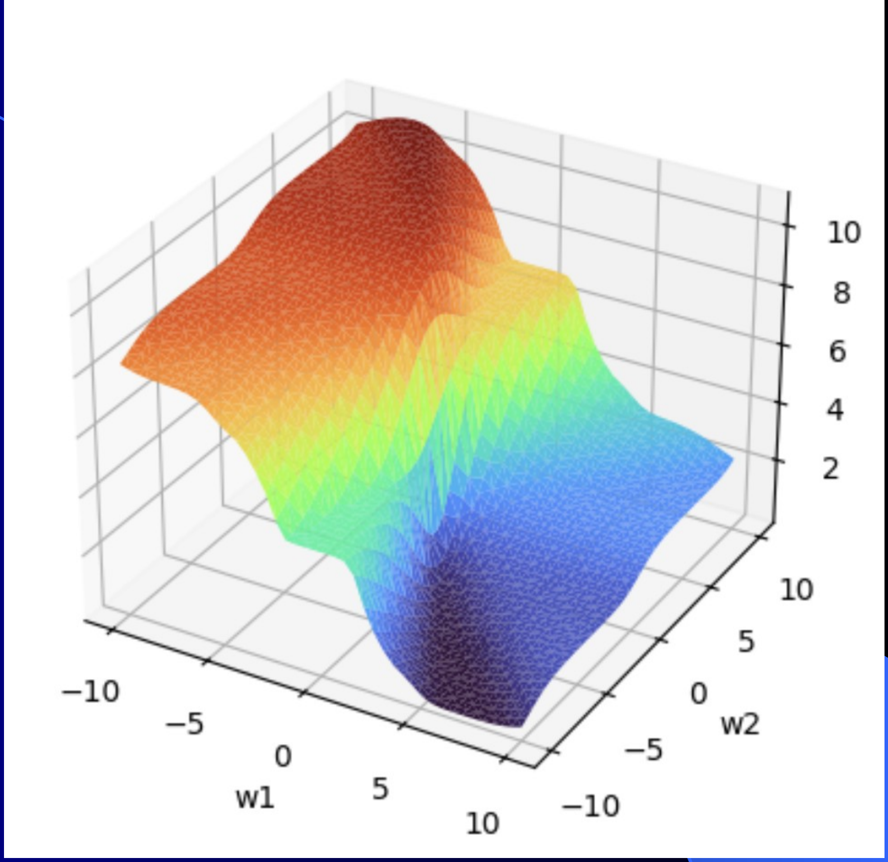


- Weights are updated even when the output would be correct
- Because this model is single layer and because of the SSE objective function, the error surface is guaranteed to be parabolic with only one minima
- Learning rate
  - If learning rate is too large can jump around global minimum
  - If too 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

Batch vs Stochastic Update

- To get the true gradient, we need to sum errors over the entire training set and only update weights at the end of each epoch
- Batch (gradient) vs stochastic (on-line, incremental)
  - SGD (Stochastic Gradient Descent)
  - With the stochastic gradient descent algorithm, you update after every pattern, just like with the perceptron algorithm (even though that means each change may not be along the true gradient)
  - Stochastic is more efficient and best to use in almost all cases, though not all have figured it out yet
  - We'll talk about this in more detail when we get to Backpropagation

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
- Single 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)

Training/Testing

Training/Testing

- Gradient descent algorithm gives a training algorithm
  - Doesn't define testing
- How do we know how good it is?
  - Are we in a local minima?

- Four methods that we commonly use:
  - Training set method
  - Static split test set
  - Random split test set CV
  - *N*-fold cross-validation
  - The last two are the more accurate approaches

Training Set Method

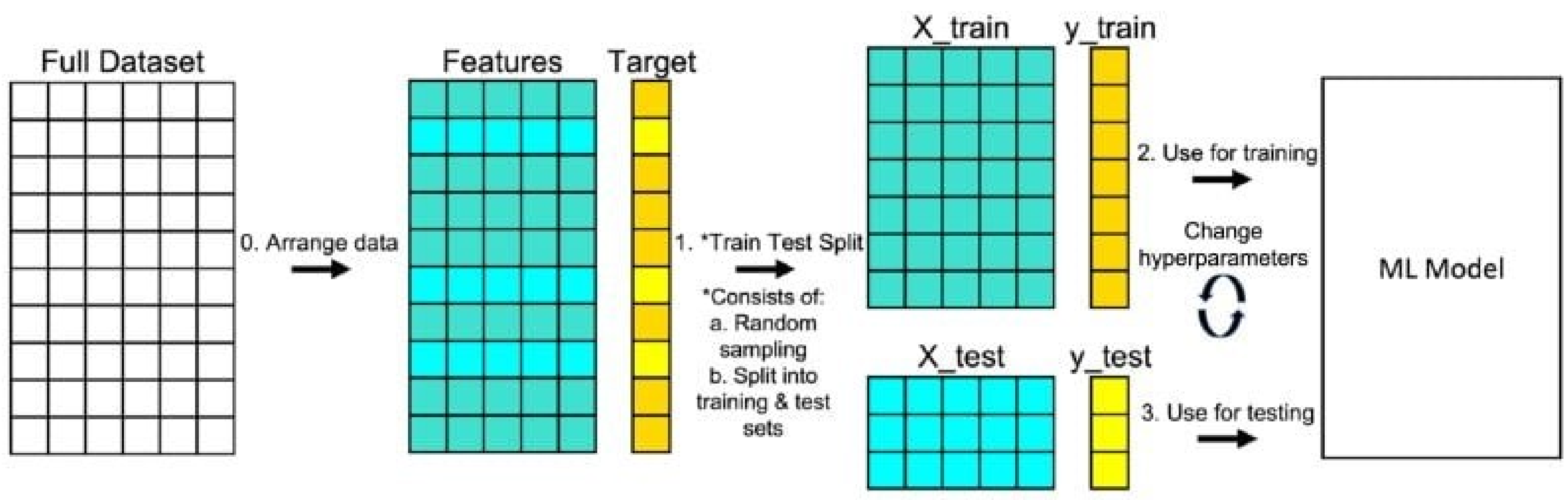
- Procedure
  - Build model from the training set
  - Compute accuracy on the same training set
- Simple but least reliable estimate of future performance on unseen data (a rote learner could score 100%!)
- Not used as a performance metric but it is often important information in understanding how a machine learning model learns
- This is information which you will often report in your labs and then compare it with how the learner does on a better method

Static Training/Test Set

- Static Split Approach
  - The data owner makes available to the machine learner two distinct datasets:
    - One is used for learning/training (i.e., inducing a model), and
    - One is used exclusively for testing
- Note that this gives you a way to do repeatable tests
- Can be used for challenges (e.g. to see how everyone does on one particular unseen set, method we use for helping grade your labs.)
- Be careful not to overfit the Test Set ("Gold Standard")

Random Training/Test Set Approach

- Random Split CV Approach (aka holdout method)
  - The data owner makes available to the machine learner a single dataset
  - The machine learner splits the dataset into a training and a test set, such that:
    - Instances are randomly assigned to either set
    - The distribution of instances (with respect to the target class) is hopefully similar in both sets due to randomizing the data before the split
      - Stratification is an option to ensure proper distribution
    - Typically 60% to 90% of instances is used for training and the remainder for testing – the more data there is the more that can be used for training and still get statistically significant test predictions
  - Useful quick estimate for computationally intensive learners
  - Not statistically optimal (high variance, <u>unless</u> lots of data)
    - Could get a lucky or unlucky test set
  - Best to do multiple training runs with different random splits. Train and test *m* different splits and then average the accuracy over the *m* runs to get a more statistically accurate prediction of generalization accuracy.


Full Dataset
0. Arrange data
Features
Target
1. *Train Test Split
*Consists of:
a. Random sampling
b. Split into training & test sets
X_train
y_train
X_test
y_test
2. Use for training
Change hyperparameters
3. Use for testing
ML Model

In [3]:
```python
url = 'https://raw.githubusercontent.com/mGalarnyk/Tutorial_Data/master/King_County/kingCountyHouseData.csv'
df = pd.read_csv(url)
# Selecting columns I am interested in
columns = ['bedrooms','bathrooms','sqft_living','sqft_lot','floors','price']
df = df.loc[:, columns]
df.head(10)
```

Out[3]:

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	price
0	3	1.00	1180	5650	1.0	221900.0
1	3	2.25	2570	7242	2.0	538000.0
2	2	1.00	770	10000	1.0	180000.0
3	4	3.00	1960	5000	1.0	604000.0
4	3	2.00	1680	8080	1.0	510000.0
5	4	4.50	5420	101930	1.0	1225000.0
6	3	2.25	1715	6819	2.0	257500.0

In [4]:
```python
features = ['bedrooms','bathrooms','sqft_living','sqft_lot','floors']
X = df.loc[:, features]
y = df.loc[:, ['price']]
```

	bedrooms	bathrooms	sqft_living	sqft_lot	floors	price
0	3	1.000000	1180	5650	1.000000	221900.000000
1	3	2.250000	2570	7242	2.000000	538000.000000
2	2	1.000000	770	10000	1.000000	180000.000000
3	4	3.000000	1960	5000	1.000000	604000.000000
4	3	2.000000	1680	8080	1.000000	510000.000000
5	4	4.500000	5420	101930	1.000000	1225000.000000
6	3	2.250000	1715	6819	2.000000	257500.000000
7	3	1.500000	1060	9711	1.000000	291850.000000
8	3	1.000000	1780	7470	1.000000	229500.000000
9	3	2.500000	1890	6560	2.000000	323000.000000

df

X = df.loc[:, features]]
y = df.loc[:, 'price']

	bedrooms	bathrooms	sqft_living	sqft_lot	floors
0	3	1.000000	1180	5650	1.000000
1	3	2.250000	2570	7242	2.000000
2	2	1.000000	770	10000	1.000000
3	4	3.000000	1960	5000	1.000000
4	3	2.000000	1680	8080	1.000000
5	4	4.500000	5420	101930	1.000000
6	3	2.250000	1715	6819	2.000000
7	3	1.500000	1060	9711	1.000000
8	3	1.000000	1780	7470	1.000000
9	3	2.500000	1890	6560	2.000000

X

	price
0	221900.000000
1	538000.000000
2	180000.000000
3	604000.000000
4	510000.000000
5	1225000.000000
6	257500.000000
7	291850.000000
8	229500.000000
9	323000.000000

y

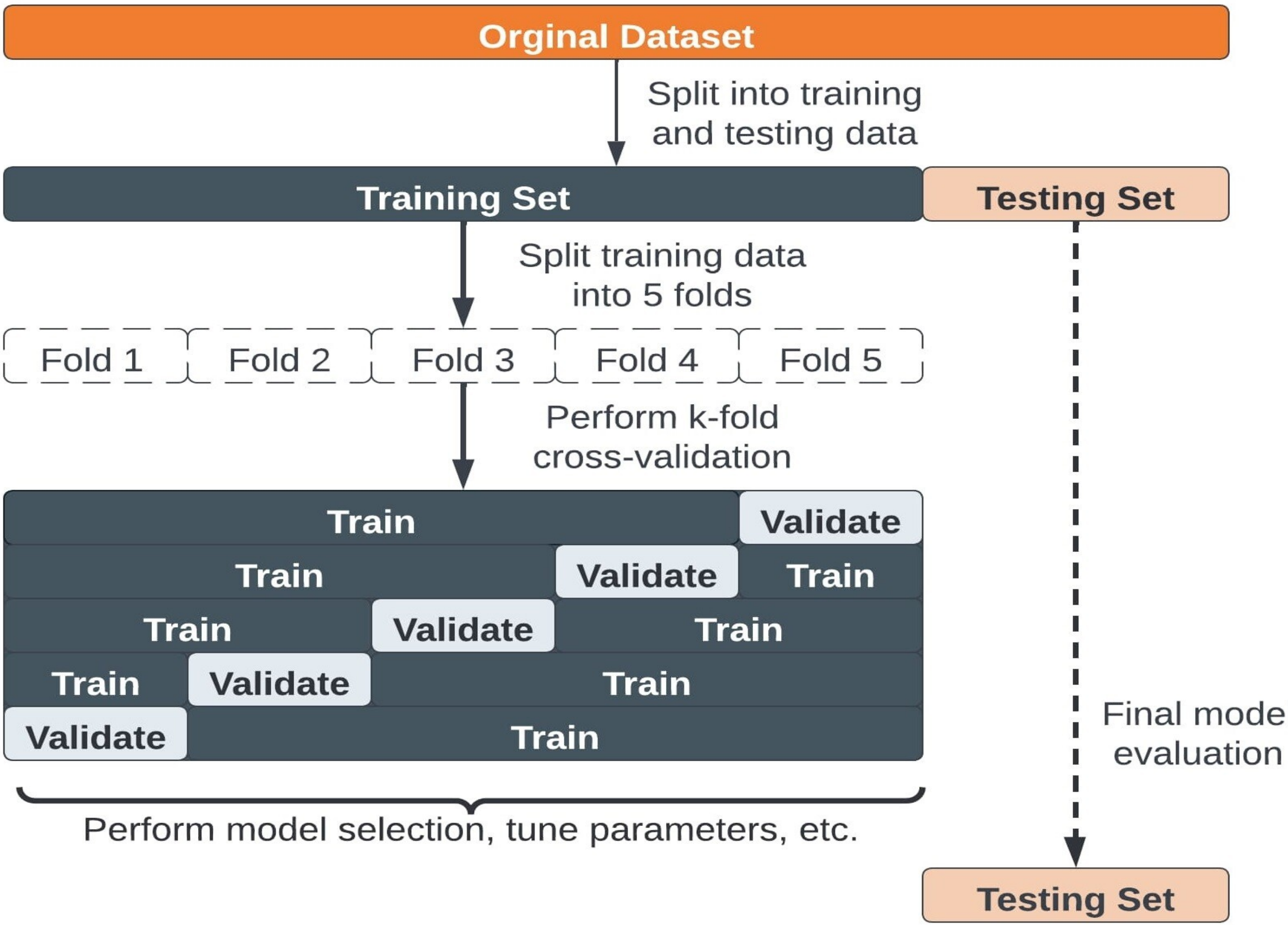
In [5]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0, train_size = .75)
```

Cross-Validation (CV)

- Cross-Validation (CV) – Cross-validation is a resampling method that uses different portions of the data to test and train a model on different iterations
- We then average the results of these iterations
- With CV we avoid having data just used for either training or test, and give all data a chance to be part of each, thus getting more accurate results
- Mainly used for comparing different learning models

N-fold Cross-validation

- Use all the data for both training and testing
  - Statistically more reliable
  - All data can be used which is good, especially for small data sets
- Procedure
  - Partition the randomized dataset (call it D) into N equally-sized subsets $S_1, \ldots, S_N$
  - For $k = 1$ to N
    - Let M_k be the model induced from $D - S_k$
    - Let a_k be the accuracy of M_k on the instances of the test fold S_k
  - Return $(a_1+a_2+\ldots+a_N)/N$



Orginal Dataset
Split into training
and testing data
Training Set
Testing Set
Split training data
into 5 folds
Fold 1
Fold 2
Fold 3
Fold 4
Fold 5
Perform k-fold
cross-validation
Train
Validate
Train
Validate
Train
Train
Validate
Train
Train
Validate
Train
Validate
Train
Perform model selection, tune parameters, etc.
Final model
evaluation
Testing Set

N-fold Cross-validation (cont.)

- The larger *N* is, the smaller the variance in the final result
- The limit case where $N = |D|$ is known as *leave-one-out CV* and provides the most reliable estimate. However, it is typically only practical for small instance sets
- Commonly, a value of N=10 is considered a reasonable compromise between time complexity and reliability
- Still must chose an actual model to use during execution – how?

N-fold Cross-validation (cont.)

- The larger *N* is, the smaller the variance in the final result
- The limit case where $N = |D|$ is known as *leave-one-out CV* and provides the most reliable estimate. However, it is typically only practical for small instance sets
- Commonly, a value of N=10 is considered a reasonable compromise between time complexity and reliability
- Still must chose an actual model to use during execution - how?
  - Could select the one model that was best on its fold?
  - All data! With any of the approaches
- Note that N-fold CV is just a better way to estimate how well we will do on novel data, rather than a way to do *model selection*

Quiz

- Password is:
  - gradient