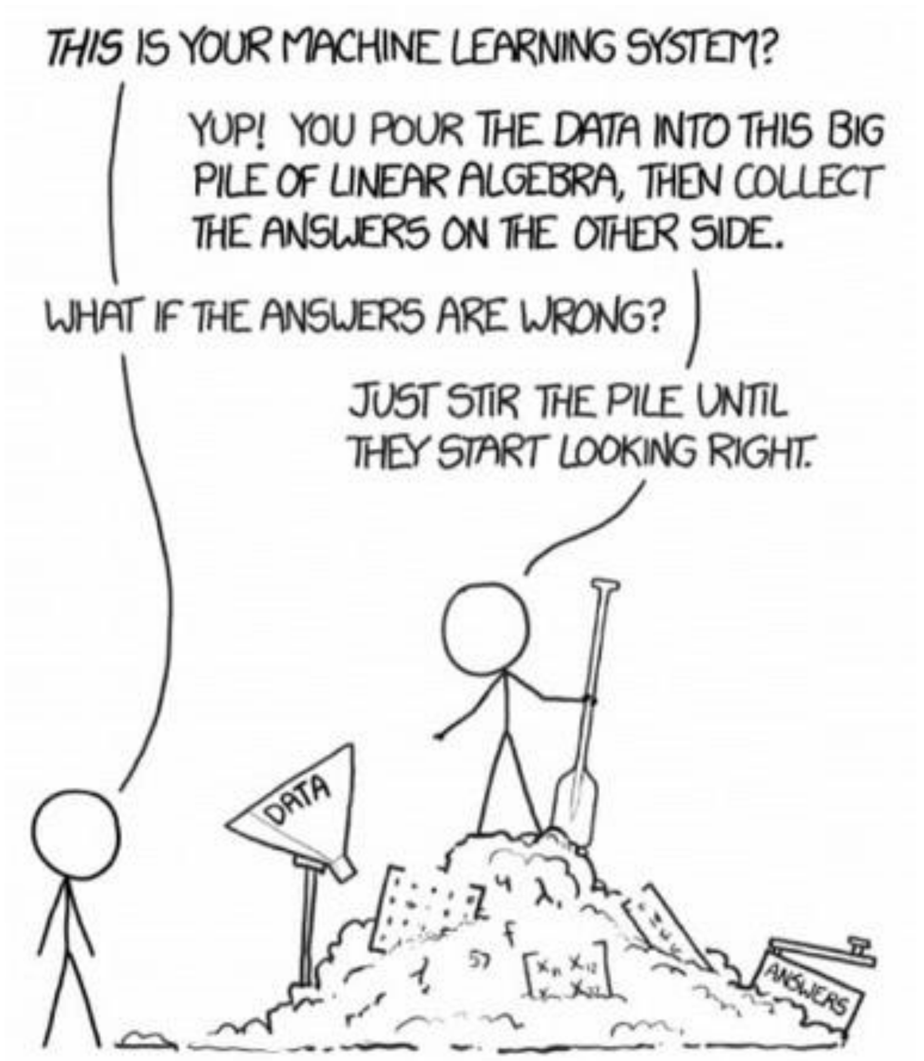
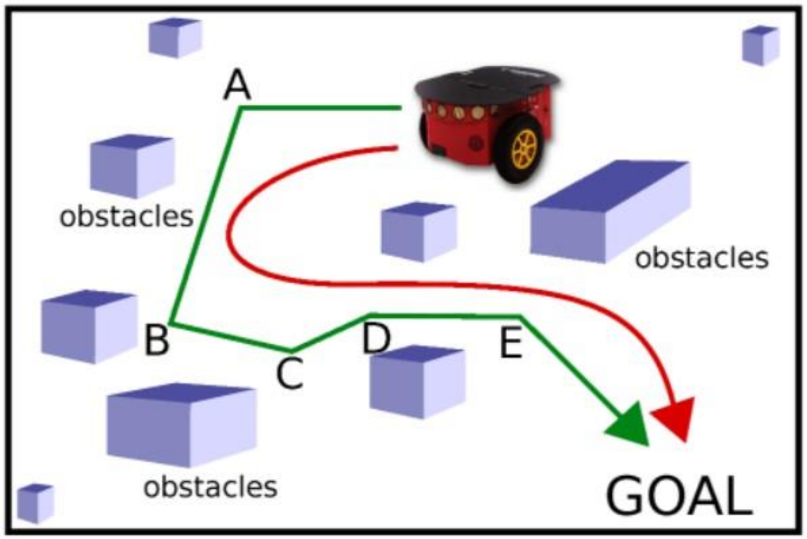

THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS

Machine Learning INTRODUCTION

Mike "Perk" Perkins
Adjunct Professor
Carnegie Mellon Africa

AI: The Rational Agent Approach

Artificial Intelligence: the study of human made *rational agents*



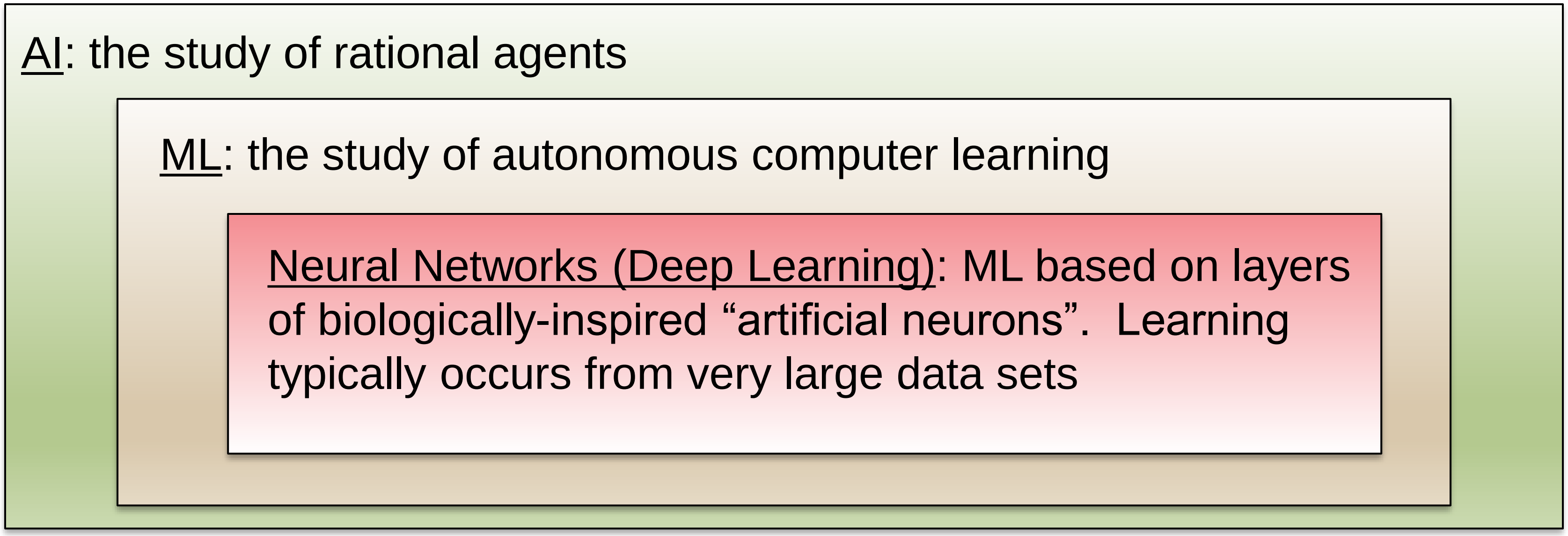
- **Agent**
  - Something that acts
    - ✓Operates autonomously
    - ✓Perceives the environment
    - ✓Adapts to change
- **Rational agent**
  - One that acts to achieve the best expected outcome in the face of uncertainty

"It has been said that man is a rational animal. All my life I have been searching for evidence which could support this" (Bertand Russell)

Machine Learning

Machine Learning: The ability of a computer to learn without being explicitly programmed

- **Major Aspects of ML include**
  - Processing data to detect and extrapolate patterns
  - Adapting as new data becomes available



AI: the study of rational agents

ML: the study of autonomous computer learning

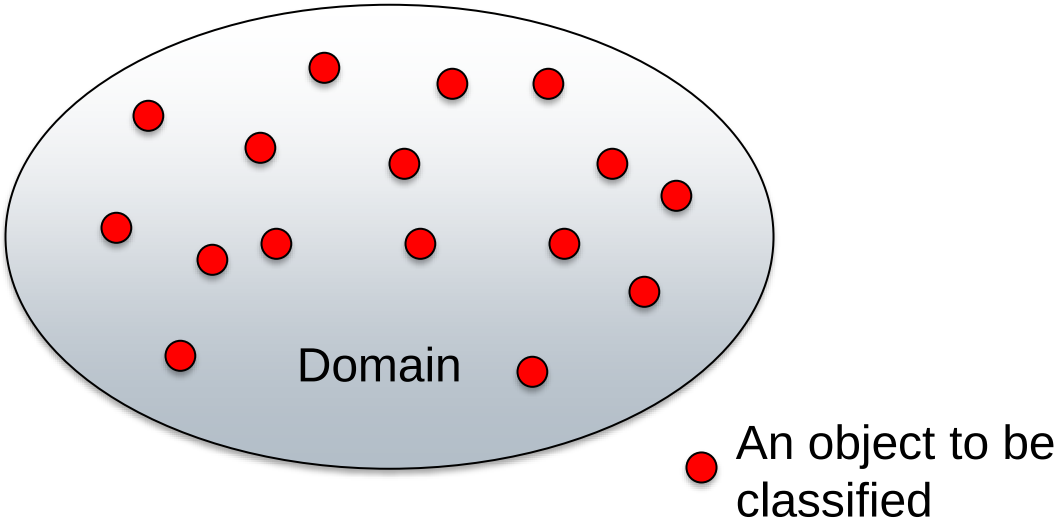
Neural Networks (Deep Learning): ML based on layers of biologically-inspired “artificial neurons”. Learning typically occurs from very large data sets

Machine Learning—Classification

Carnegie Mellon Africa

- **Consider a set of objects that we wish to classify**
  - Example: the set could be the passengers on the Titanic
  - Potential passenger feature vector
    - ✓ (Gender, Age, FareClass, Deck)

Who survived?



We'll call the set of objects to classify the domain

Binary Classification

- **Assign every domain object to one of two classes**
  - Titanic passengers either *survived* or *perished*
- **Other domains and binary classification tasks**
  - Images of Faces
    - ✓ classify as *male* or *female*, *old* or *young*, etc.
  - Emails
    - ✓ classify as *spam* or *ham, personal* or *work,* etc.
  - Customer financial records
    - ✓ classify customer as *likely* or *unlikely* to default on a loan
  - Medical assessments
    - ✓ classify patient as *sick* or *healthy*

Multi-class Classification

- **Objects can be assigned to more than two classes**
- **Some domains and multi-class classification tasks**
  - Images of Faces
    - ✓ classify as *young-male*, *young-female*, *old-male*, *old-female*
  - Images of hand-written numerals
    - ✓ classify as the numbers *{0, 1, 2, ..., 9}*
  - Traffic-related measurements
    - ✓ classify commute time as *short*, *average*, or *long*

Extracting Features from Objects

- **The objects to classify are often very complex**
- **Simplification**
  - Analyze the objects to extract a set of <u>features</u>
  - Group the features together into a <u>feature vector</u>
- **We refer to a feature vector as an <u>instance</u>**
  - We call the set of all possible feature vectors the <u>instance space</u>

Example Titanic feature vector:
(Gender, Age, FareClass, Deck)

Feature Vector Example

- **Problem—classify people as *male* or *female***
- **Components of feature vector**
  - Height, weight, length of hair, length of nose, length of foot
  - (H, W, L_h, L_n, L_f)
- **The instance space**
  - The set of all possible feature vectors for a human

In this example the instance space is infinite

Typical Case

- **Somehow we obtain a labeled set of actual feature vectors**
  - *Labeled* means we know the class for each feature vector
  - We call this set the *Training Set*
- **Example**
  - $(H, W, L_h, L_n, L_f) = (2\text{m}, 75\text{Kg}, 7\text{cm}, 4\text{cm}, 20\text{cm}) \rightarrow \textit{Male}$
  - $(H, W, L_h, L_n, L_f) = (1.6\text{m}, 50\text{Kg}, 15\text{cm}, 3\text{cm}, 16\text{cm}) \rightarrow \textit{Female}$
  - *etc.*

feature vectors labels

The Training Set

- **The training set is a set of ordered pairs**

$$\mathcal{T} = \{(\underline{\mathbf{x}}_i, l(\underline{\mathbf{x}}_i))\}$$

- **Here $\underline{x}_i$ is an instance (feature vector) and *l( )* is the <u>true</u> labeling function**
- **The training set is <u>samples</u> of the true classification function we want to learn**
  - The training set does NOT contain every vector from the instance space (which is often infinite)
- **Ideally, the training set is large**

Supervised Learning

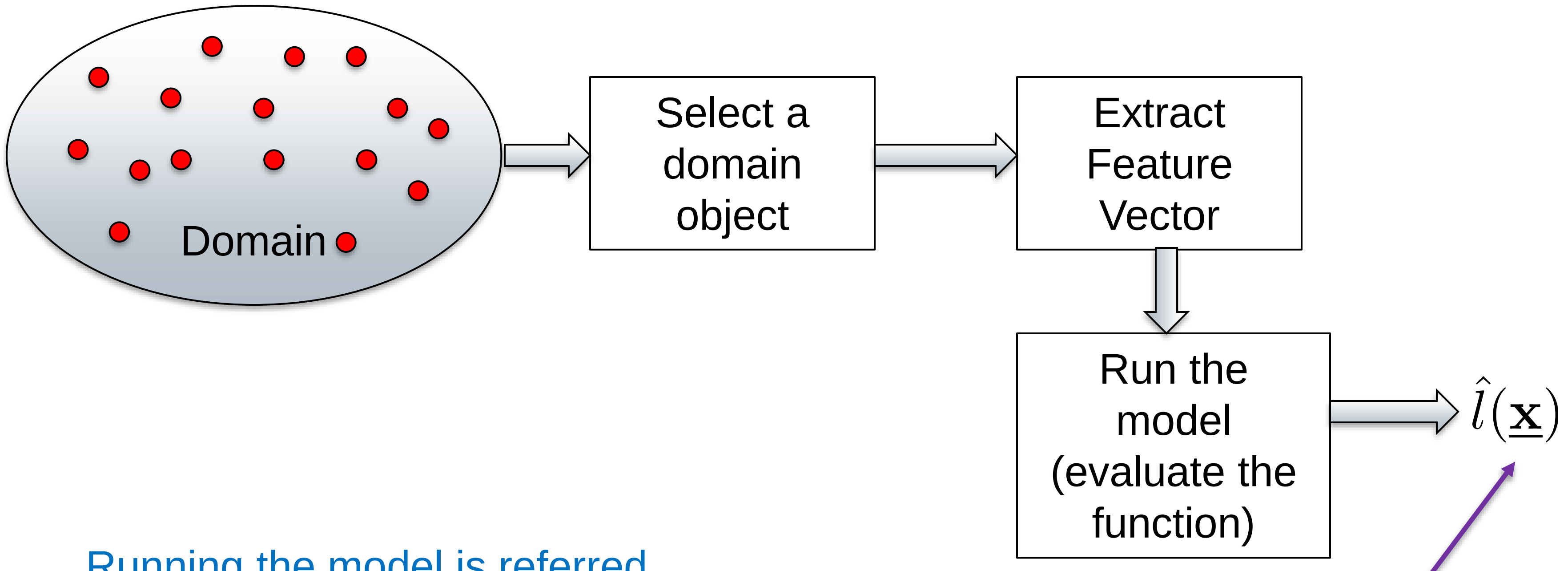
Supervised learning is using a labeled training set to learn a <u>function</u> for mapping instances to classes

- The function learned, $\hat{l}(\underline{\mathbf{x}})$, is called a *model*
- The function learned can be used to *predict* the class of instances that are <u>not</u> in the training set
- Hopefully,

$$\hat{l}(\underline{\mathbf{x}}) \approx l(\underline{\mathbf{x}})$$

Diagram of system's use after training

- **An object from the domain is selected by the system**
  - Example: a music clip is captured by a microphone

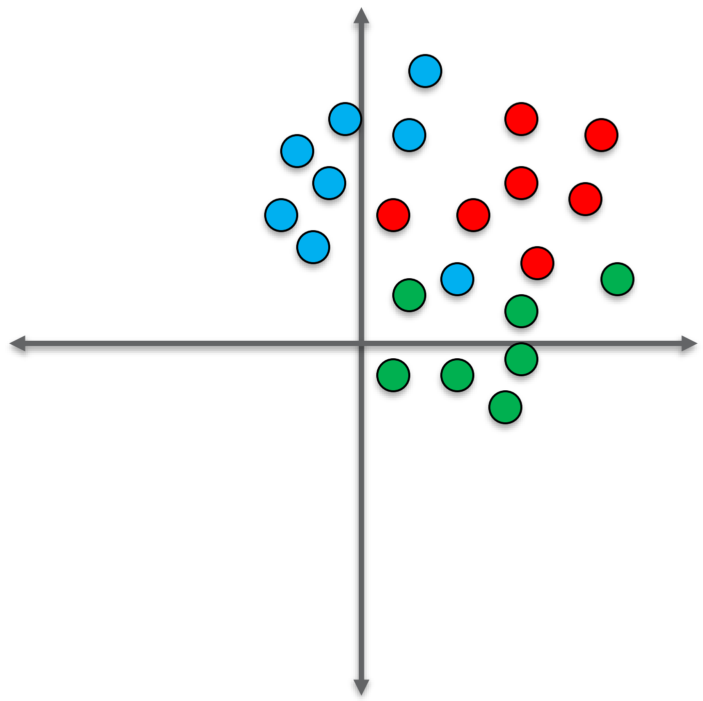

Domain
Select a domain object
Extract Feature Vector
Run the model (evaluate the function)
$\hat{l}(\underline{\mathbf{x}})$

Running the model is referred to as performing an *ML inference*

The predicted genre (Jazz, Rock, Classical, ...)

Visualizing performance

- **Assume the feature vector is two dimensional (2D) and there are 3 classes.**
  - The points below represent instances in the instance space (red: class 1, green: class 2, blue: class 3)



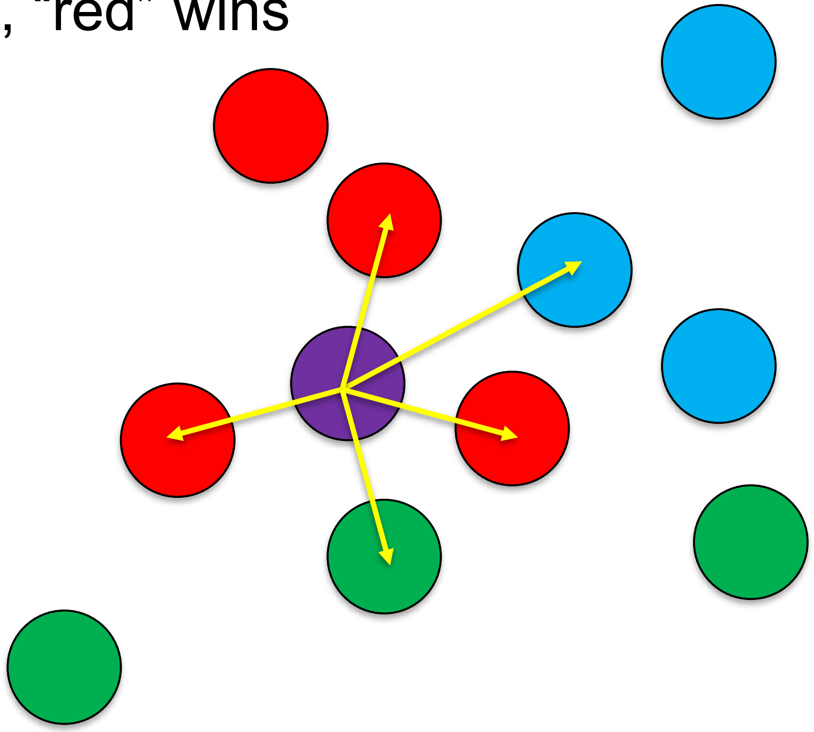
These points correspond to feature vectors

NOTE: The mapping of domain objects to feature vectors may **not** be 1-to-1 (invertible)

K-nearest neighbor classification

- **To classify the unknown purple object we look at the classes of its *k* nearest neighbors (e.g. *k=5*) and take a majority vote**
  - In this example, “red” wins

This algorithm requires the notion of a *distance* between feature vectors



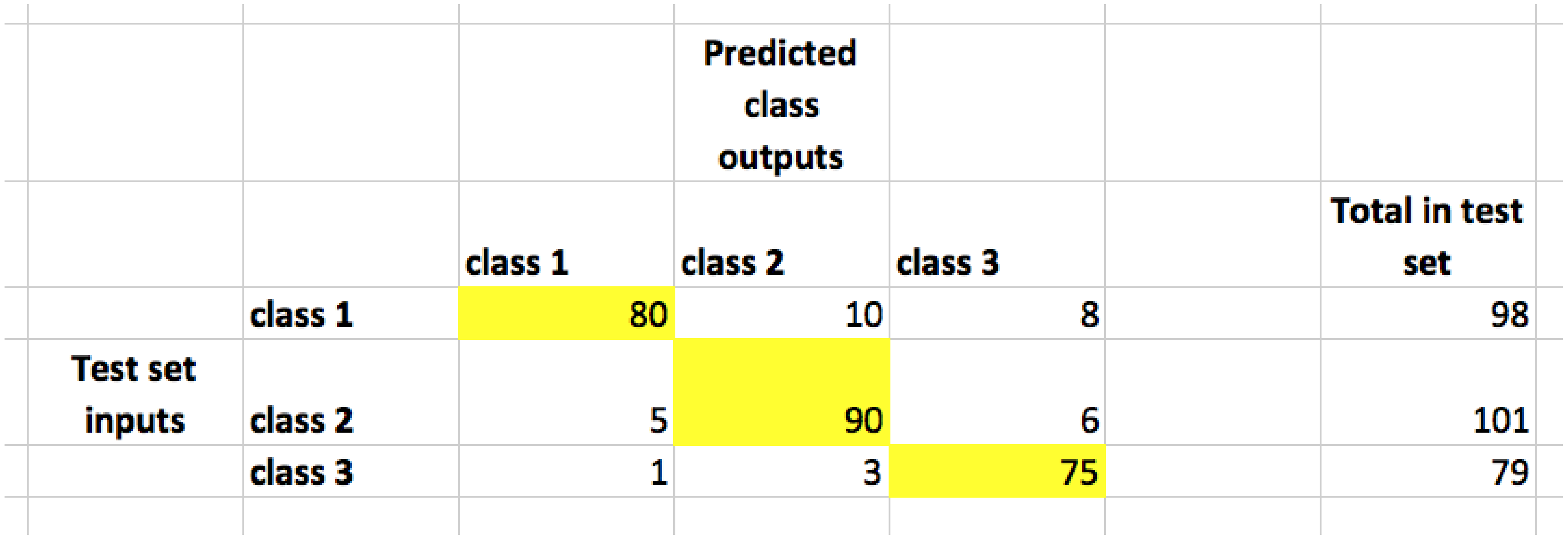
K-nearest neighbor classification

- **Armed with a "distance" the following algorithm is sometimes practical**
  - To classify instance ***x***
    - ✓ Compute the distance between ***x*** and each element of the training set
    - ✓ Determine its *k* nearest neighbors in the training set (*k* can be 1)
    - ✓ Assign to ***x*** the majority class of its *k*-nearest neighbors

Assessing Classification Performance

Carnegie Mellon Africa

- **Create a confusion matrix**
  - Use the matrix to estimate useful probabilities



			Predicted class outputs			
		class 1	class 2	class 3		Total in test set
	class 1	80	10	8		98
Test set inputs	class 2	5	90	6		101
	class 3	1	3	75		79

  - Note: the highlighted values represent correct classifications

Assessing Performance

- **Accuracy**
  - Estimates the probability

$$P(\hat{l}(\underline{\mathbf{x}}) = l(\underline{\mathbf{x}}))$$

✓ Add the diagonal entries of the matrix and divide by the sum of all entries. For the previous matrix we get 245/278 = 88%

- **Error rate**
  - Estimates the probability

$$P(\hat{l}(\underline{\mathbf{x}}) \neq l(\underline{\mathbf{x}}))$$

✓ One minus the accuracy. In our example, 33/278 = 12%

CAUTION! In an unbalanced environment, the accuracy may be meaningless. You might get high accuracy just by always outputting the most probable value!

Assessing Performance

- **Terminology to describe a model's performance *__with respect to a given class__*:**
  - True Positive (TP): A *correctly* accepted instance of the class
  - False Positive (FP): An *incorrectly* accepted non-instance of the class
  - True Negative (TN): A *correctly* rejected non-instance of the class
    - ✓ However, the rejected instance may or may not be correctly classified with respect to the remaining classes!
  - False Negative (FN): An *incorrectly* rejected instance of the class

Assessing Performance

- **Precision**
  - For a given class, it estimates the percentage of positive class identifications that are actually members of that class

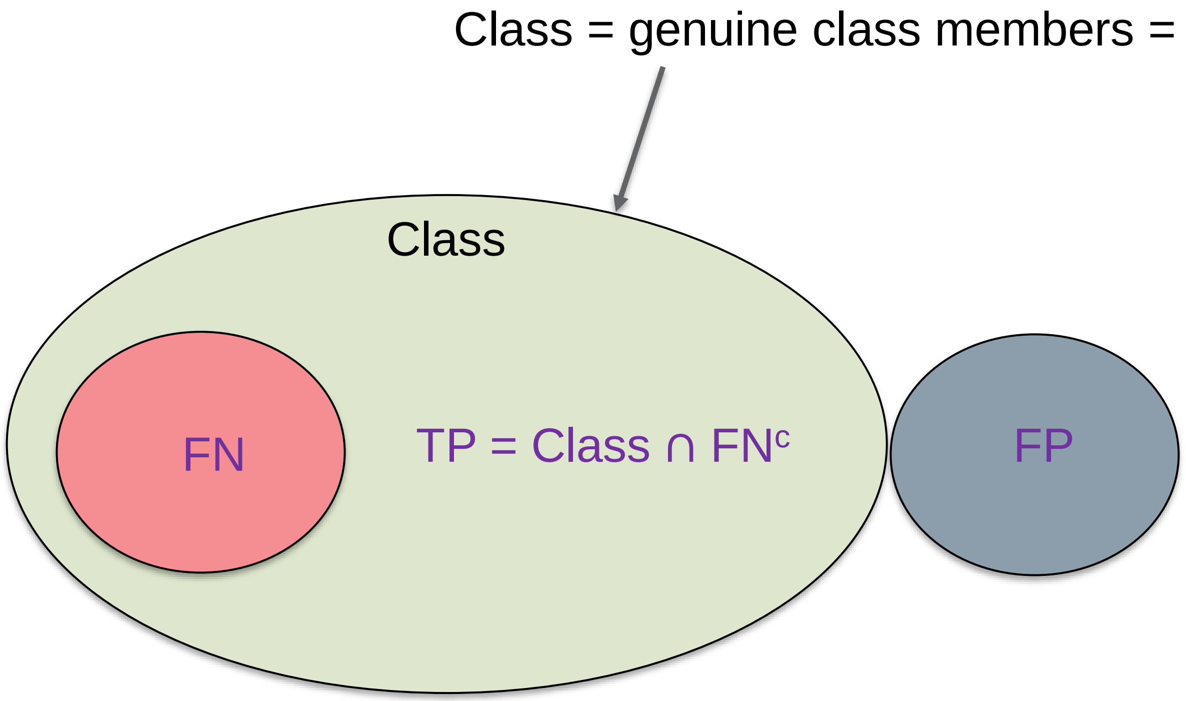
$$Precision = \frac{TP}{TP + FP}$$

- **Recall**
  - For a given class, it estimates the percentage of that class's members that will be correctly classified

$$Recall = \frac{TP}{TP + FN}$$

These metrics are often more useful than accuracy and error rate for an unbalanced training set

Visualizing Precision and Recall



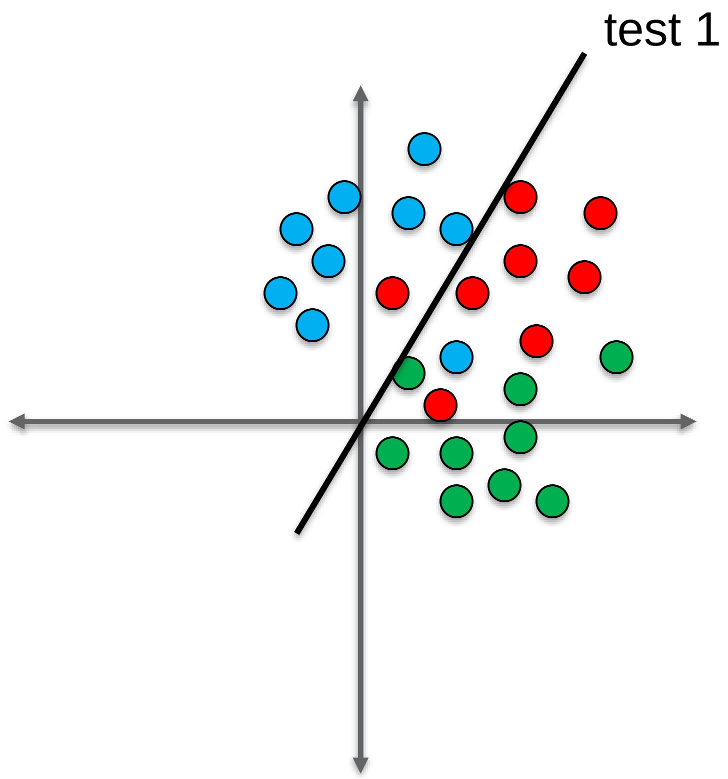
> The model classifies TP ∪ FP as "in class"
> Both FN and FP are model errors!

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

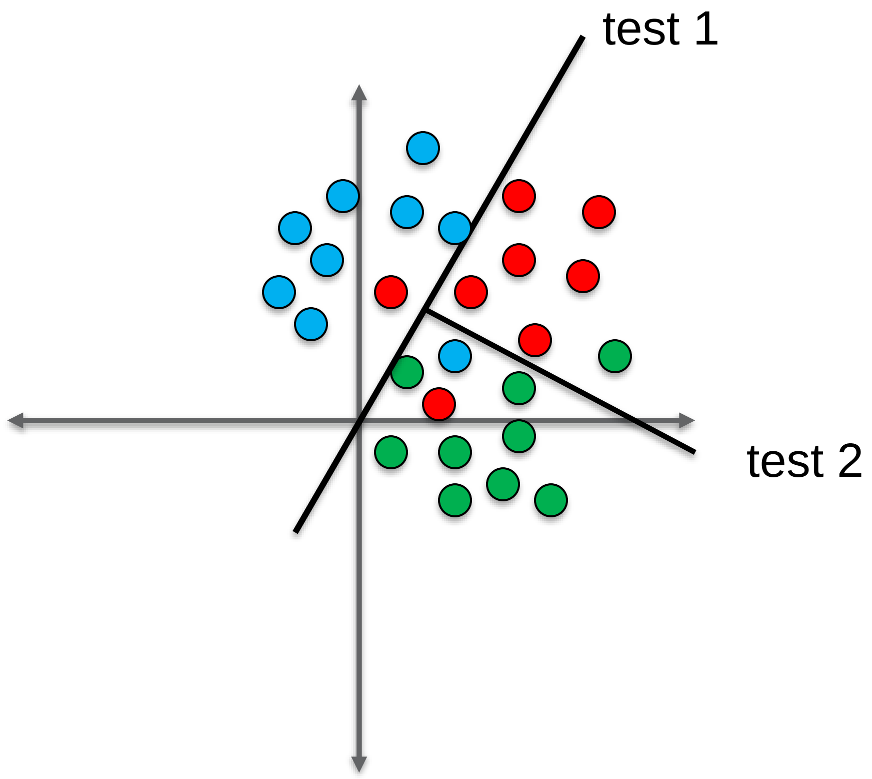
Separating points in instance space

- **One approach—learn lines that separate instances into classes**
  - First, separate one class from the rest



Separating the points

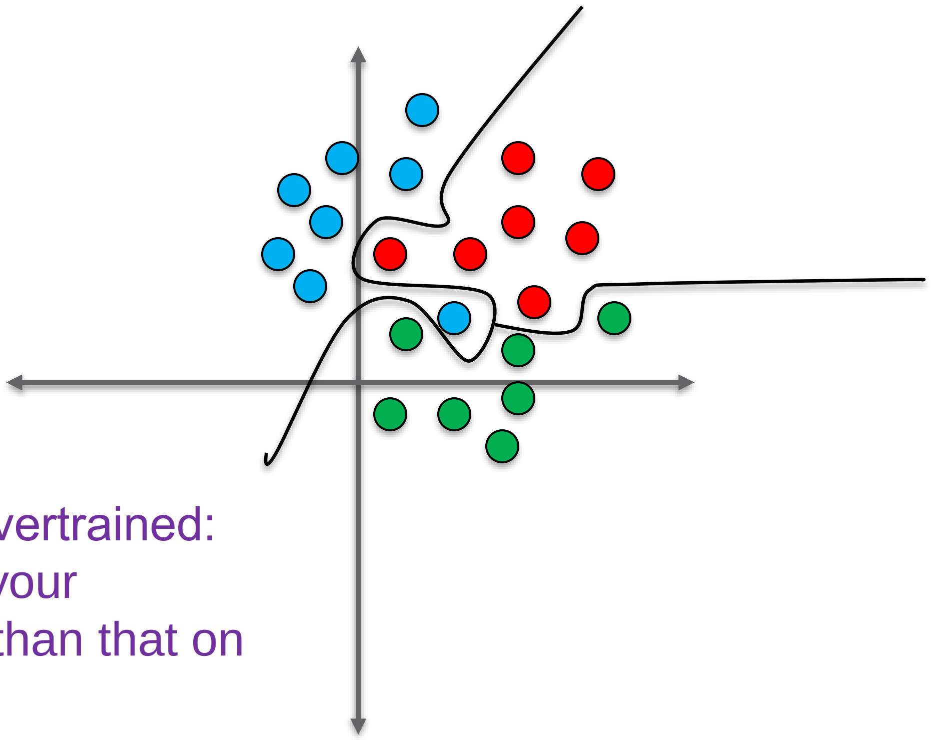
- **Second: separate the mixed class into two classes**



- "Support Vector Machines" (SVMs) can be used to find a line that separates two classes

Overtraining

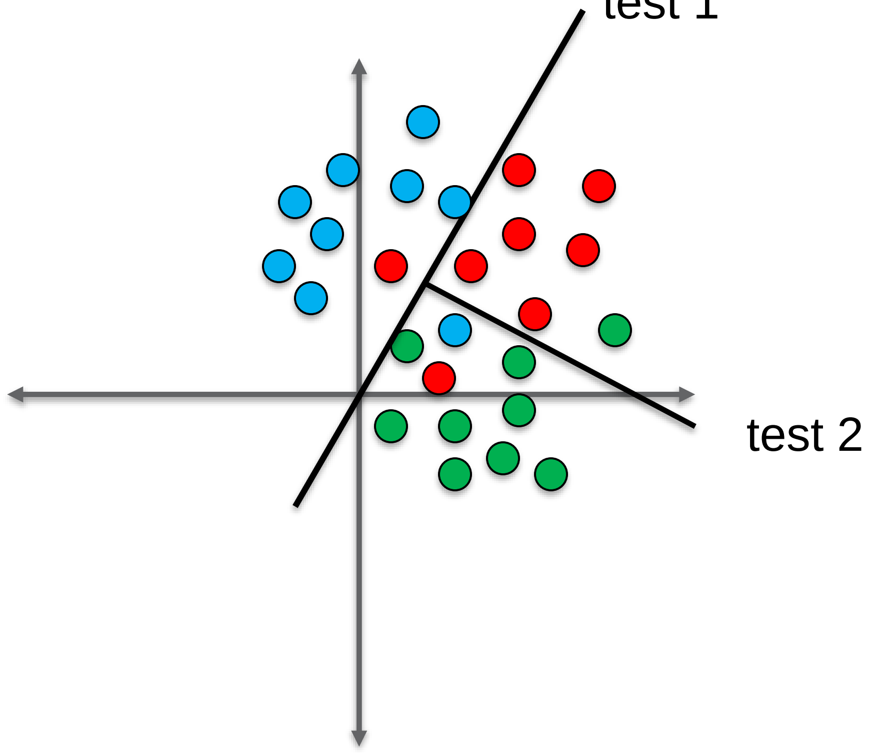
- **Danger—adapting your function too strongly to your training set**
  - Trying too hard to solve the outlier problem may yield a bad solution that doesn't generalize



One way to test if you've overtrained: see if the performance on your training set is much better than that on a separate test set

What if the feature vector is 3D or higher?

- **In higher dimensional spaces we learn a *hyperplane* not a line**



Conceptually, there is no difference!

Using the data wisely

- **Given a single labeled set of data, how should one proceed?**
  - One option
    - ✓ Shuffle the data
    - ✓ Divide it into a training set and a test set
      - – Divisions of 70% / 30% and 80% / 20% are common
    - ✓ Learn the classification function using the training set
    - ✓ Assess the model's performance using the test set
  - Another more complicated option is k-fold cross validation

Neural Networks

“A single neuron in the brain is an incredibly complex machine that even today we don't understand. A single 'neuron' in a neural network is an incredibly simple mathematical function that captures a minuscule fraction of the complexity of a biological neuron.” (Andrew Ng, AI Guru)

Regression

- **Sometimes the goal is to assign a real number to an instance**
- **We call this type of prediction regression, and typically differentiate it from classification**
  - Example
    - ✓ Given the earlier person feature vector (H, W, L_h, L_n, L_f)
      - (Height, Weight, Length_hair, Length_nose, Length_foot)
    - ✓ Predict the *interocular_distance* (the distance between the eyes)

NN's are capable of regression and classification

Using Regression to Classify

- **Assume the feature vectors are of dimension *N***
- **Assume there are C classes {1, 2, ..., C}**
- **Assign the integer *i* to every instance of class *i***
  - Use the TS to learn a real-valued function, *f*

$$f\colon \mathbb{R}^N \longmapsto \mathbb{R}$$

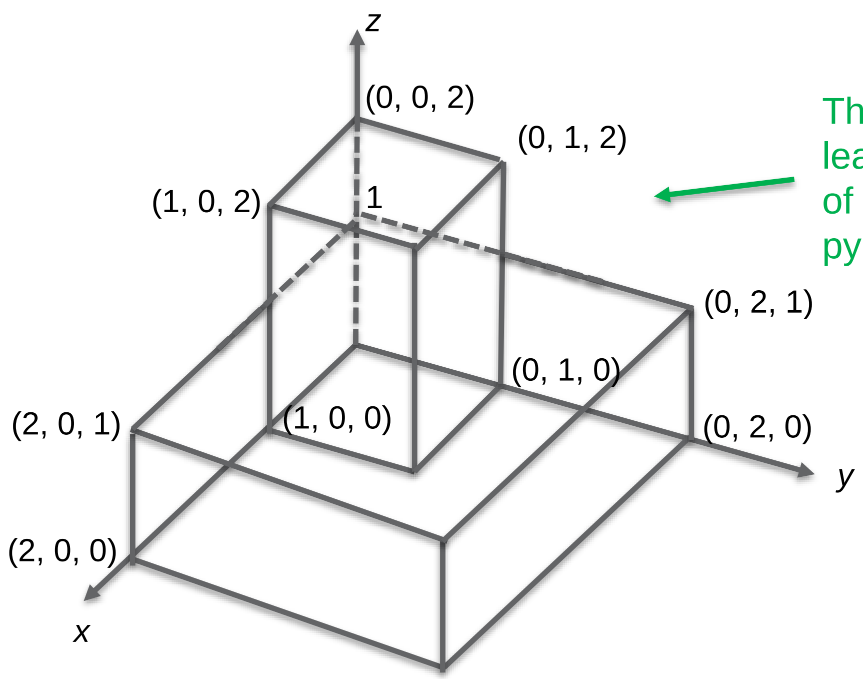
  - Roundoff the real output to an integer in $\{1, 2, \ldots C\}$

Using Regression to Classify

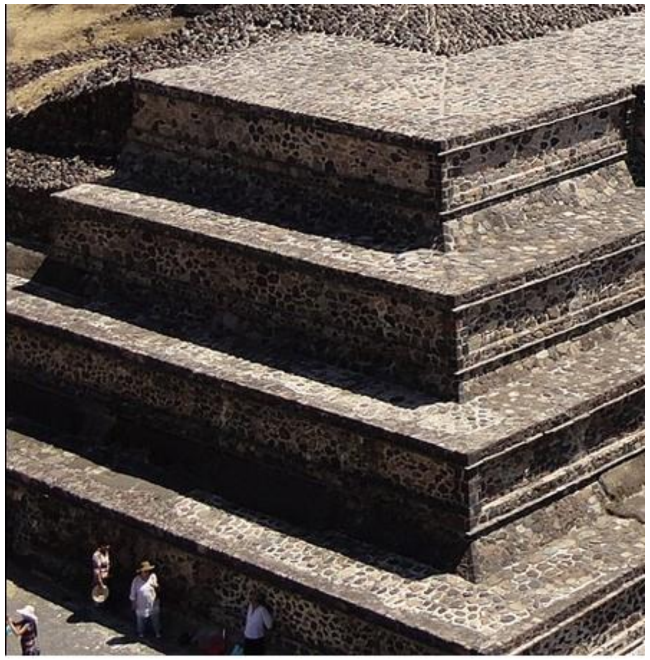
Carnegie Mellon Africa

- **Example (binary classification)**

  Assume:
  - 2D feature vectors *(x,y)*
  - All class two instances lie inside the square of size one in the *(x,y)* plane
  - All class one instances lie inside the square of size two and outside the square of size one



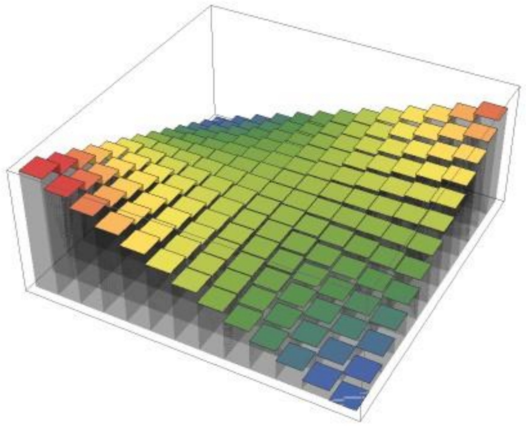
The regression should learn the 3D surface of a simple 2-step pyramid



4-step pyramid

Using Regression to Classify

- **What's the point?**
  - It can be shown that any continuous function can be approximated by a suitable NN
    - ✓ Classifier step functions can be approximated by continuous functions
      - – Result: arbitrarily complex classifiers can be learned by an NN
        - » So we can treat classification as a regression problem on one variable



But this approach is not used in practice!

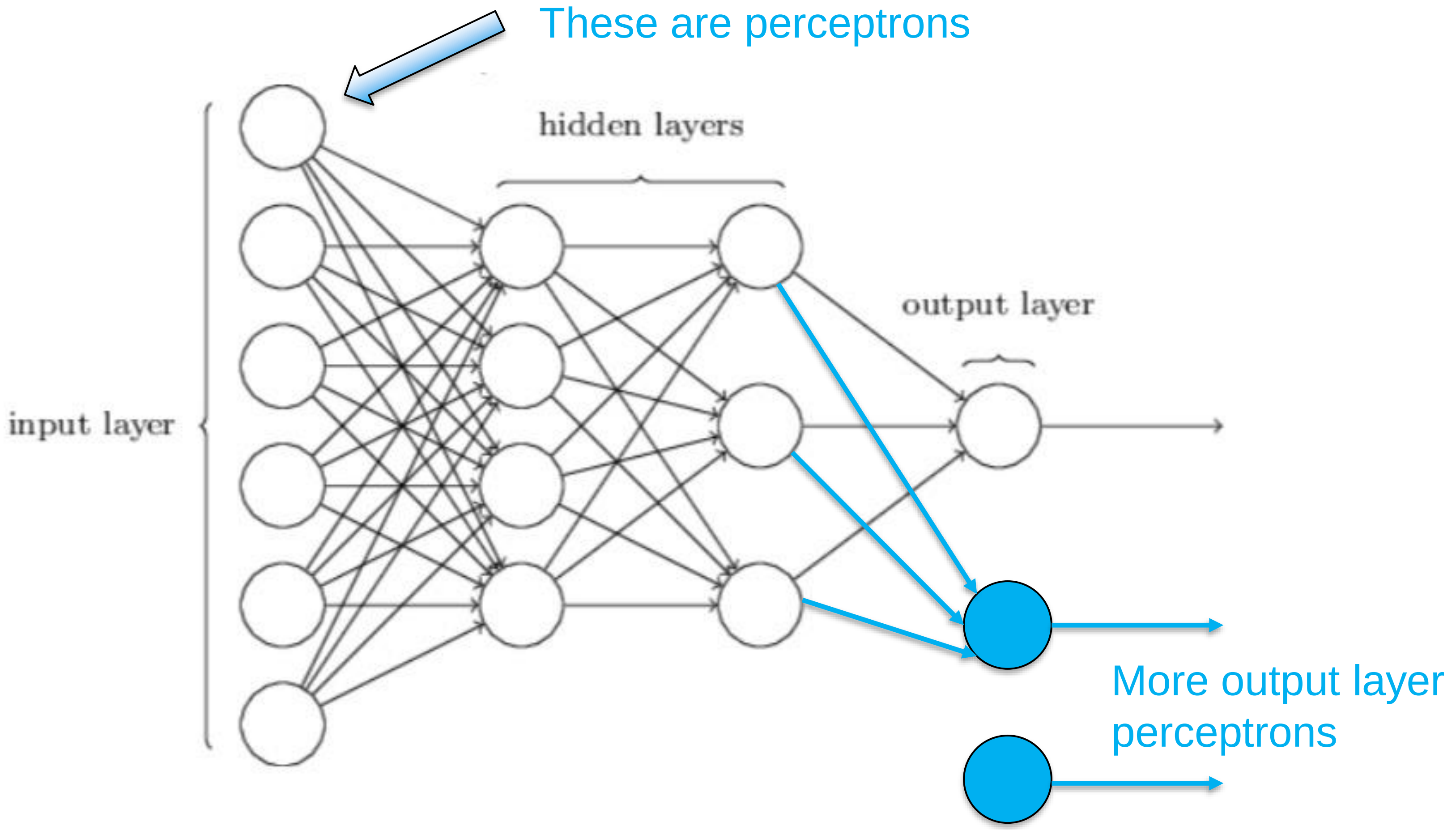
A Better Approach

- **Learn *C* functions total, one for each class**
- **The learning goal for each class's function is...**
  - ...take the value 1 on domain instances that are in the class
  - ...take the value 0 on domain instances that are not in the class
- **Another perspective**
  - The NN learns <u>one function</u> from *N*-dimensional space to *C*-dimensional space

$$f\colon\ \mathbb{R}^N \longmapsto \mathbb{R}^C$$

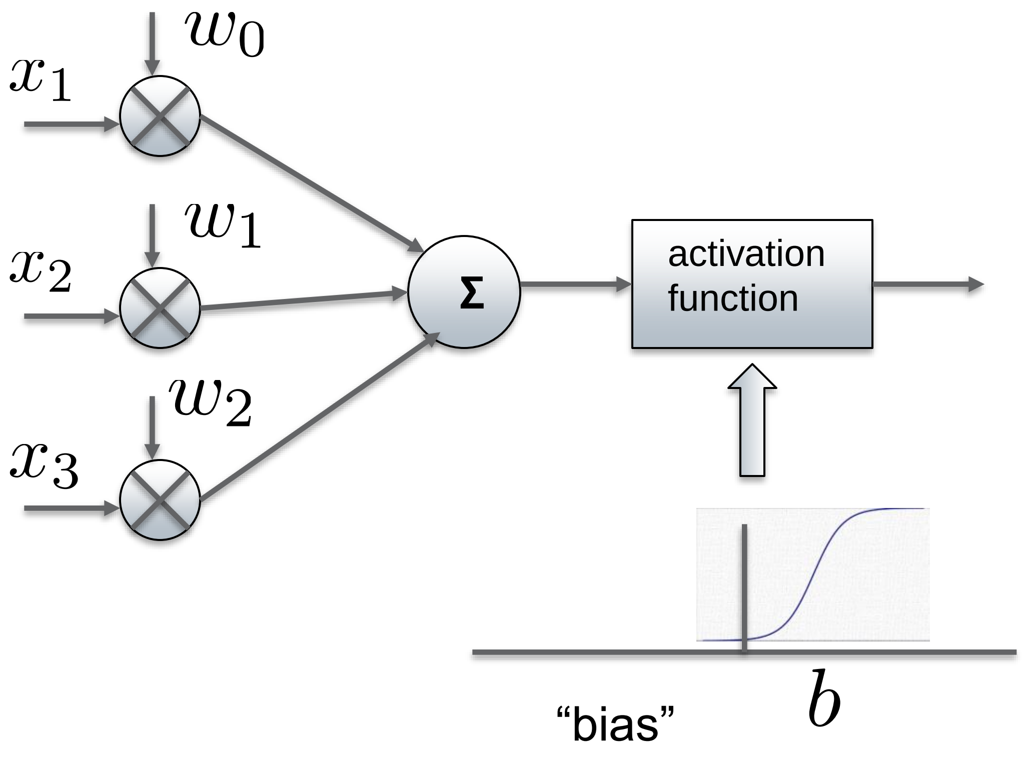
Neural Network Diagram

- Neural networks are built by connecting a basic unit, perceptrons, together



The Perceptron's calculation

- **A perceptron computes the dot product of its input vector, x, with its weights, w, and thresholds the result with respect to a bias, *b***



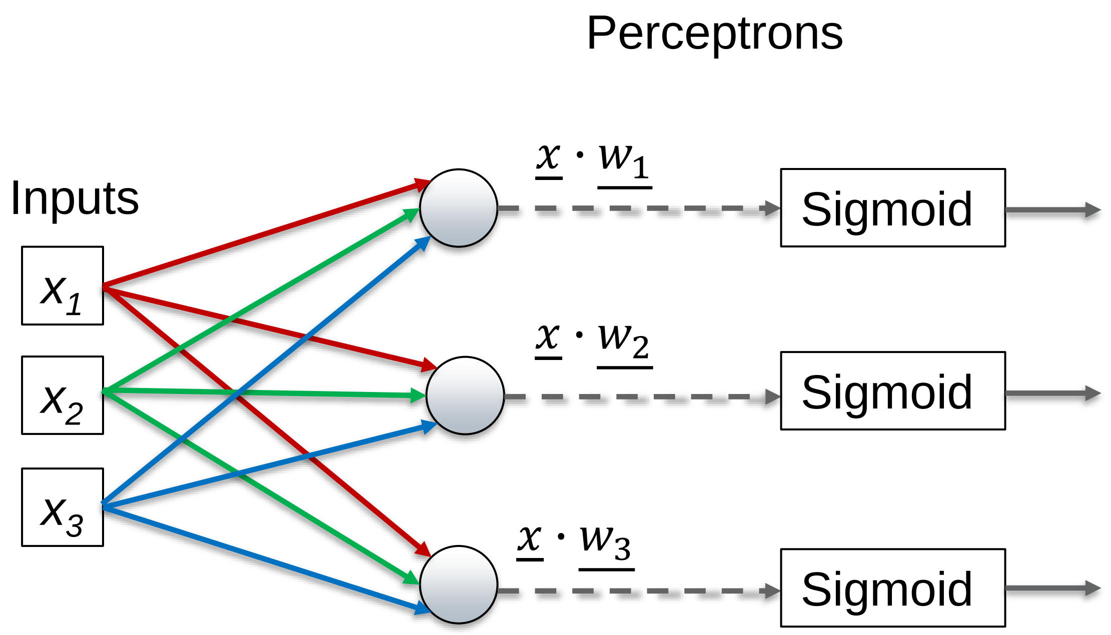
Threshold curve is often sigmoid in shape

Neural Networks for classification

- **Perceptrons are the neurons of neural networks**
- **Each perceptron has parameters**
  - Input weights
  - bias
- **The output of a perceptron can be an input to many downstream perceptrons**
- **The perceptrons' parameters are optimized by training the network**
  - This is how the network learns

Dot Products and Perceptron Inputs

Carnegie Mellon Africa



Perceptrons
Inputs
x_1
x_2
x_3
$\underline{x} \cdot \underline{w_1}$
$\underline{x} \cdot \underline{w_2}$
$\underline{x} \cdot \underline{w_3}$
Sigmoid
Sigmoid
Sigmoid

Here's a compact representation of these 3 dot products

$$\begin{pmatrix} \underline{w_1^T} \\ \underline{w_2^T} \\ \underline{w_3^T} \end{pmatrix} \underline{x}$$

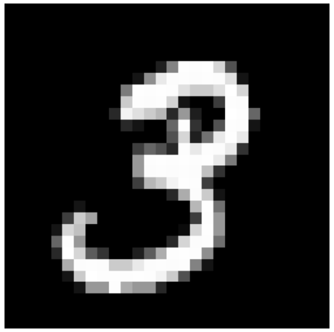
Matrix notation and linear algebra are core ML math tools

Training Set

- **Assume an object's class, *y*, will be predicted using a feature vector, *x̲***
- **The training set (whose instances are indexed by *i*)**

$$\mathcal{T} = \{(x_{i1}, x_{i2}, \ldots, x_{iN}, y_i)\}$$

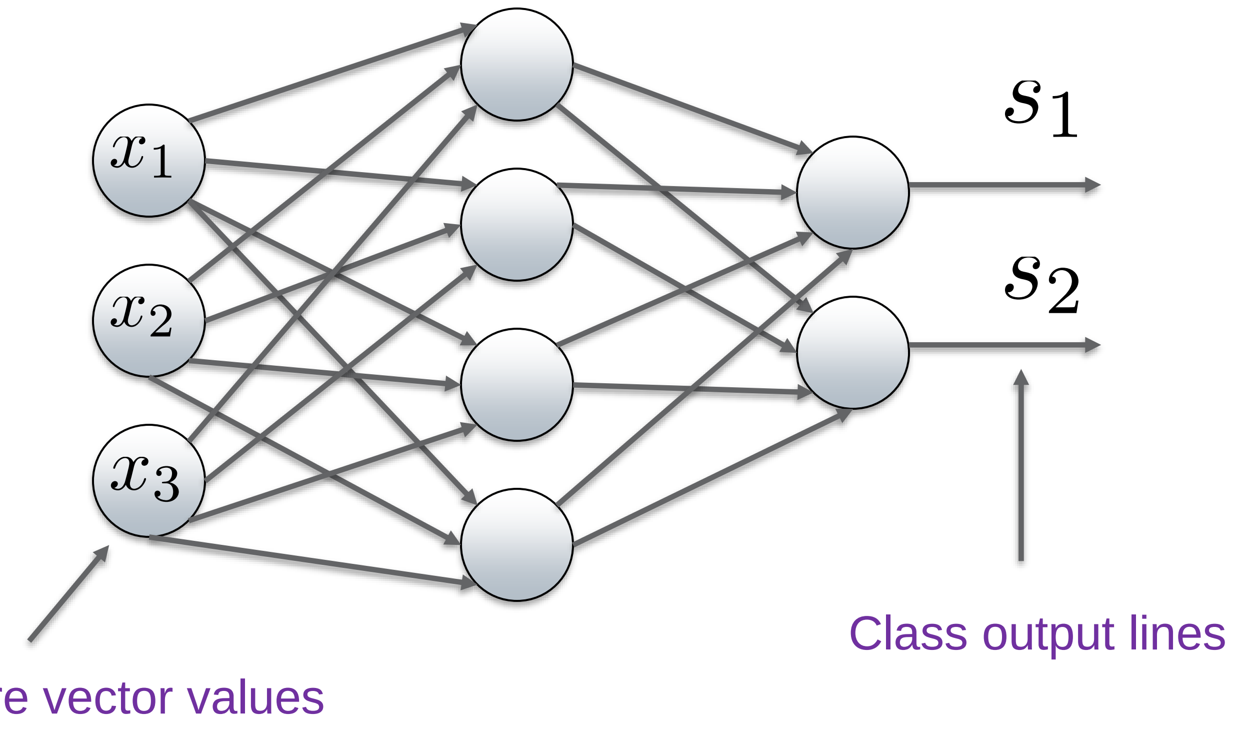
Each instance vector has *N* features



For an image, each feature might be a pixel intensity

Neural Network Classification

- **There is one output per class**
  - The largest output value indicates the predicted class

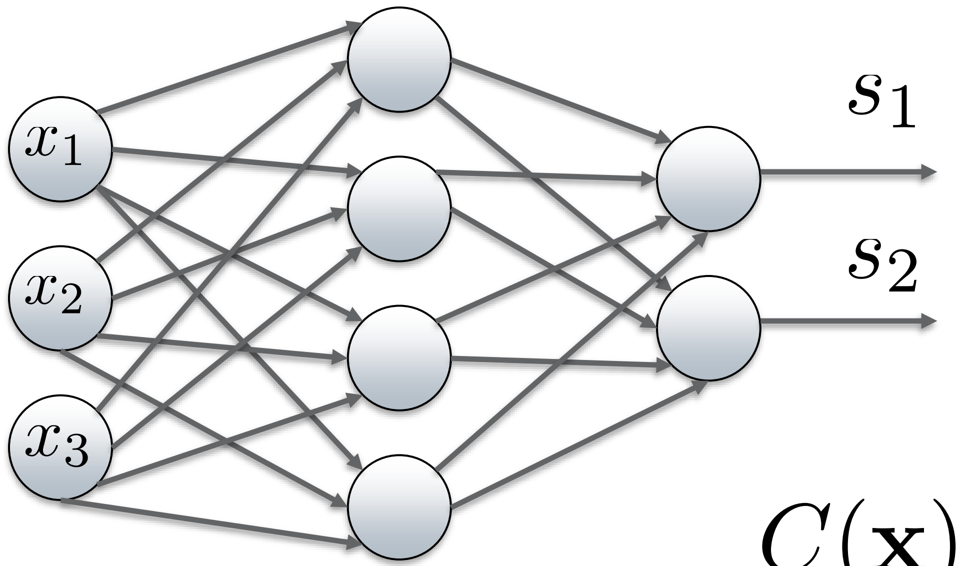


Neural network training overview

- **The perceptrons' weights and biases are iteratively adjusted using the training sequence**
  - Many passes through the training sequence are necessary
- **A gradient descent algorithm is often used to find good weights and biases**
  - We are searching through a high-dimensional space for a good parameter set
- **The back-propagation algorithm is used to compute the gradient**

A possible cost function for optimization

- **The network has one output for each class**
  - The largest output value indicates the predicted class



Let the network's output when $\underline{\boldsymbol{x}}$ is applied be $\underline{\mathbf{s}}(\underline{\boldsymbol{x}})$

Let $\underline{l}(\underline{\boldsymbol{x}})$ be a vector with 1 in the position of the true class

A possible cost function for $\underline{\boldsymbol{x}}$ is

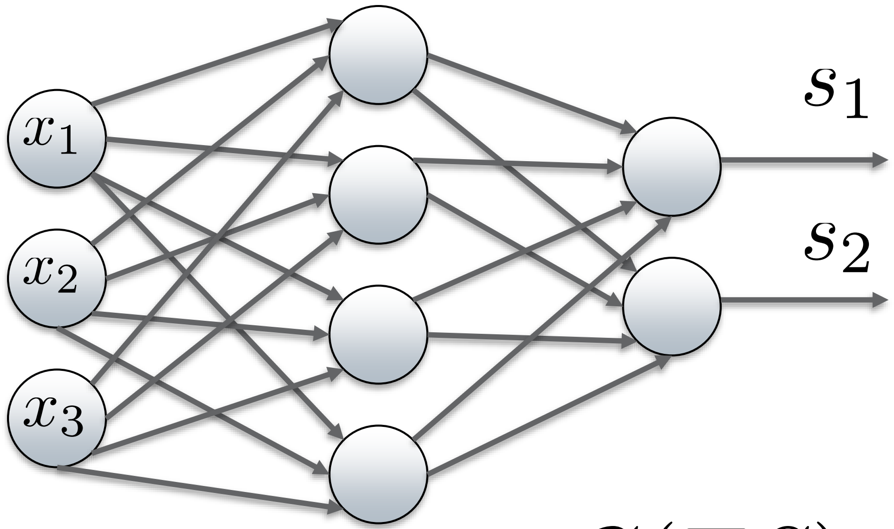
$$C(\underline{\mathbf{x}}) = \|\underline{\mathbf{l}}(\underline{\mathbf{x}}) - \underline{\mathbf{s}}(\underline{\mathbf{x}})\|^2$$

In the example above, for instances in class one, $\underline{\boldsymbol{l}}(\underline{\boldsymbol{x}}) = (1, 0)$ and for instances in class two, $\underline{\boldsymbol{l}}(\underline{\boldsymbol{x}}) = (0, 1)$

Cost function continued...

Carnegie Mellon Africa

- **To obtain the overall cost function we sum over all instances in the TS**

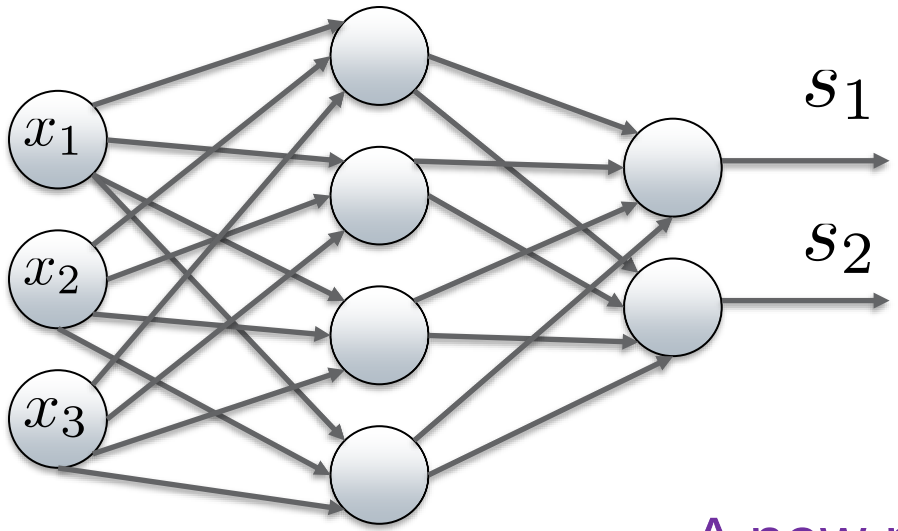


$$C(TS) = \sum_{\underline{\mathbf{x}}} \left\| \underline{\mathbf{l}}(\underline{\mathbf{x}}) - \underline{\mathbf{s}}(\underline{\mathbf{x}}) \right\|^2$$

We seek a set of weights and biases to minimize C(TS)

A better cost function

- **We can normalize the class scores first to get an output vector that looks like probabilities**



Define

Softmax function

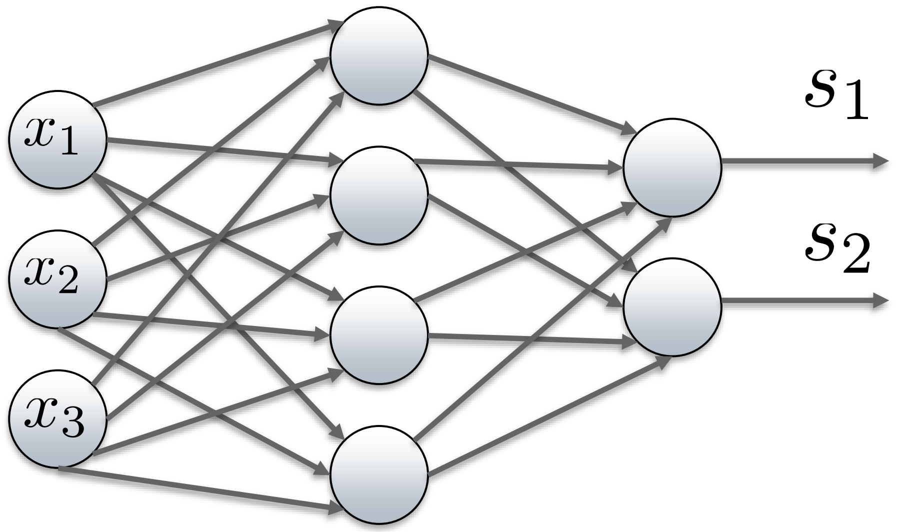
$$p_k = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

A new possible cost function

$$C(TS) = \sum_{\underline{\mathbf{x}}} \left\| \underline{\mathbf{l}}(\underline{\mathbf{x}}) - \underline{\mathbf{p}}(\underline{\mathbf{s}}) \right\|^2$$

Another cost function!

A commonly used cost function, based on maximum likelihood estimation, is the cross entropy (it involves logarithms)

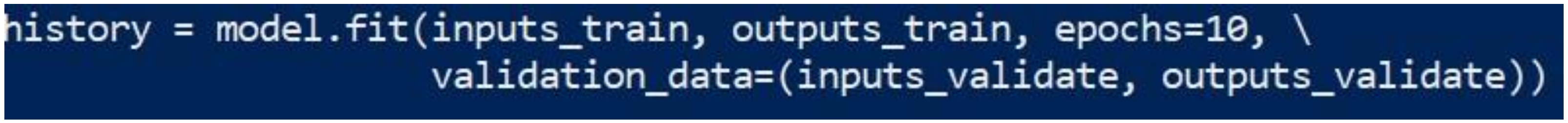


TensorFlow / Keras Concepts

- **Training a neural network classifier involves iterative processing of the training set**

One complete pass through the training set is called an <u>epoch</u>

The number of training epochs is a user-definable parameter



```
history = model.fit(inputs_train, outputs_train, epochs=10, \
                    validation_data=(inputs_validate, outputs_validate))
```

Model Parameter Training

- **Performance issue**
  - If the entire TS is processed before making an adjustment to the model's parameters, training can be very slow
- **Solution**
  - Process only a *batch* of instances from the TS before updating the model

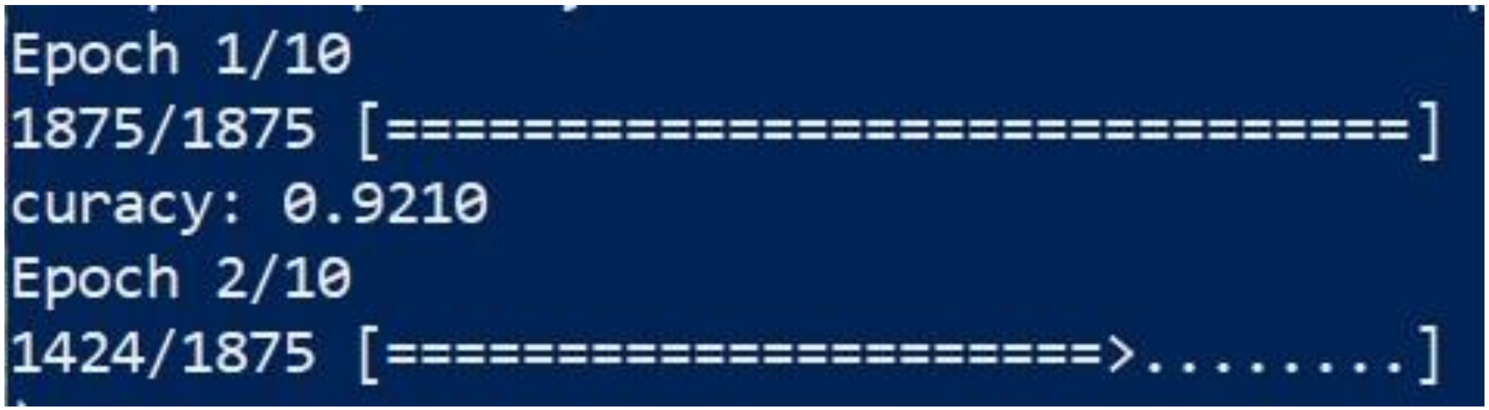
TensorFlow / Keras Concepts

- **Epochs are broken up into mini batches (or simply batches)**
- **Mini-batch size**
  - Defaults to 32
  - Can be explicitly controlled if desired

Model updates are performed after processing one mini-batch of instances from the TS

TensorFlow / Keras Concepts

- **Epoch and mini-batch view during training**



```
Epoch 1/10
1875/1875 [==============================]
curacy: 0.9210
Epoch 2/10
1424/1875 [======================>.......]
```

In this example, there are 1875 mini batches (1875*32 = 60,000 instances in TS)

Current mini-batch being processed is indicated on screen (in this case 1424)

Current epoch and total number of desired epochs are also displayed

Loss function and metric

- **Training searches for network parameters to (hopefully) minimize some *loss* function**
  - The parameters are the perceptron weights and biases
  - No guarantee a global minimum will be found!
  - Different loss functions are used
- **Model performance is assessed using some *metric***
  - metric: “*a system or standard of measurement*”
  - The loss function and the metric can be different
    - ✓ Example:
      - – Loss function: “cross entropy”
      - – Metric: “accuracy”

Why May Loss and Metric Differ?

- **Consider the accuracy of a classifier**
  - Two models may have the same accuracy when trained on the same TS
  - One model may still be better w.r.t the loss function!
    - ✓ We would expect the lower loss model to generalize better to instances outside the training set
  - The desired metric may also be discontinuous and hard/impossible to differentiate
    - ✓ Gradient descent needs derivatives

Eventually the Model is Deployed!

- **The compute resolution of the ML inference engine may be important! It might impact...**
  - ...the *final error* in a regression
  - ...the *performance* of a classification
    - ✓ What if fewer bits are used in every computation?
  - ...the *memory* required by the inference engine
    - ✓ Fewer bits per number means less storage is needed for parameters
  - ...the *speed* of performing an ML inference
    - ✓ e.g. integer vs floating point arithmetic
  - ...the *power consumption* of the inference engine
    - ✓ less computation consumes less energy

Welcome to the world of *model quantization* and *TinyML*

A Final Caution About Learning

“In our reasonings concerning matter of fact, there are all imaginable degrees of assurance, from the highest certainty to the lowest species of moral evidence. <u>A wise man, therefore, proportions his belief to the evidence</u>”

David Hume, 18th century philosopher

Reference Material

Convolutional Neural Networks for Machine Vision

“Circumstantial evidence is a very tricky thing. It may seem to point very straight to one thing, but if you shift your own point of view a little, you may find it pointing in an equally uncompromising manner to something entirely different” (Arthur Conan Doyle, 19th Century creator of Sherlock Holmes)

CNNs at 100k Feet

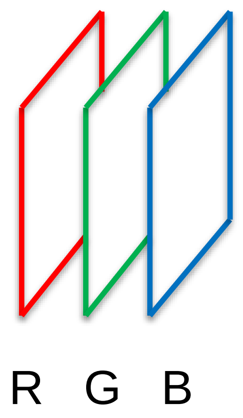
- **CNNs are very similar to ordinary neural networks**
  - Processing occurs in layers
  - Some layers are perceptron based and have learnable parameters
  - The final layer produces class scores which are fed to a real-valued loss function
  - Training is via gradient descent
- **So what's different?**
  - Some new *non-parameterized* layer types are used
  - We do *NOT* fully connect all layers
- **History: first successful applications were in the 1990s, e.g. to read postal codes**

CNN Setup

- **Assume the input image has R rows, C columns, and D color planes**
  - We'll assume D = 3 (i.e. RGB)
- **Refer to R and C as the spatial dimensions, and D as the depth dimension**
- **Think of each CNN layer as processing an input volume into an output volume**
- **CNN Layer types**
  - Input
  - Convolution
  - RELU (Rectified Linear Unit)
  - Pooling
  - Fully Connected

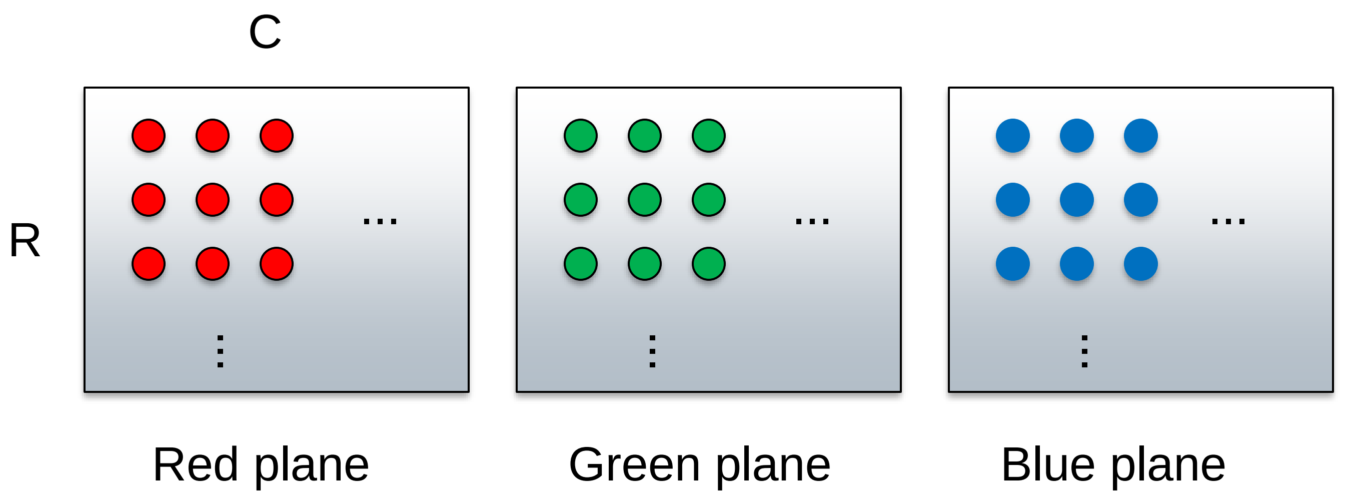
Input Layer

- **Imagine the input image as a set of three 2-D planes stacked side-by-side**
  - One plane for each color channel, e.g., RGB
- **There is one input layer perceptron for each RGB pixel**
  - Each perceptron's output is a pixel intensity (R, G, or B)
- **The input layer is therefore a 3-D volume of perceptrons**



Input Layer Unstacked

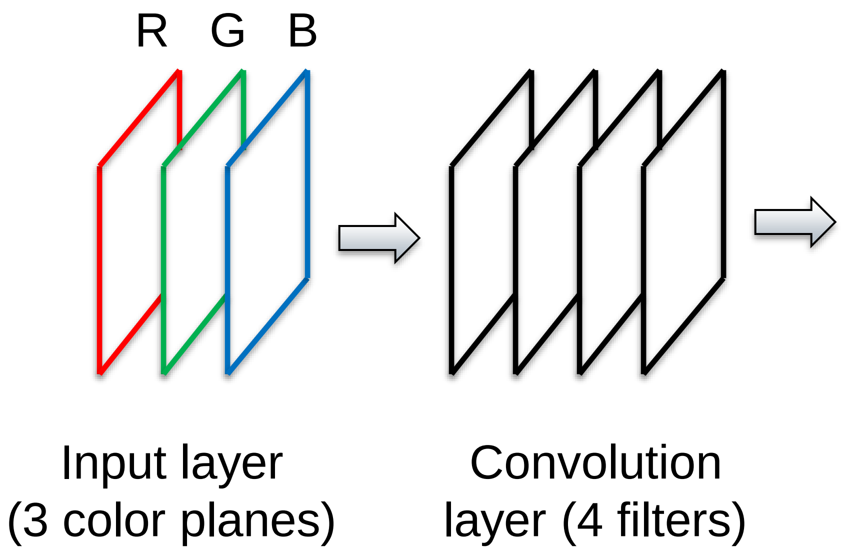
- **A view of the perceptrons in the RGB color planes, one perceptron per pixel**



Unstacked input layer planes

Convolution Layer

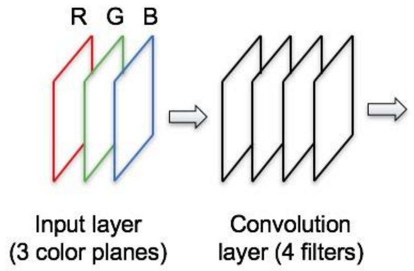
- **A convolution layer is a 3D volume of perceptrons**
  - The activation function of these perceptrons is *f(x)* = *x*
    - ✓ *Contrast this with a sigmoid. f(x) = x allows a convolution to be computed*
- **A convolutional layer comprises multiple processing planes**
  - There is one plane for each filter being trained



Convolution Layer

- **The perceptrons in a convolutional layer plane are NOT fully-connected to each pixel in the preceding layer**
  - Each connects only to a limited co-located spatial region for example, 3x3xD or 5x5xD
    - ✓ Note that connection is made to all the perceptrons in the depth dimension (e.g. to all the color planes in the input layer)
- **The perceptron weights and biases are trainable, but...**

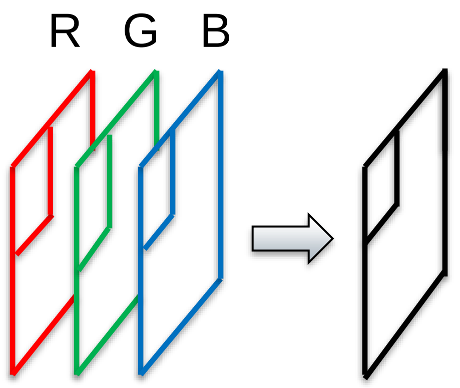
ALL perceptrons in the same filter plane use IDENTICAL parameters



The activation function of a perceptron in a convolutional layer is $f(x) = x$

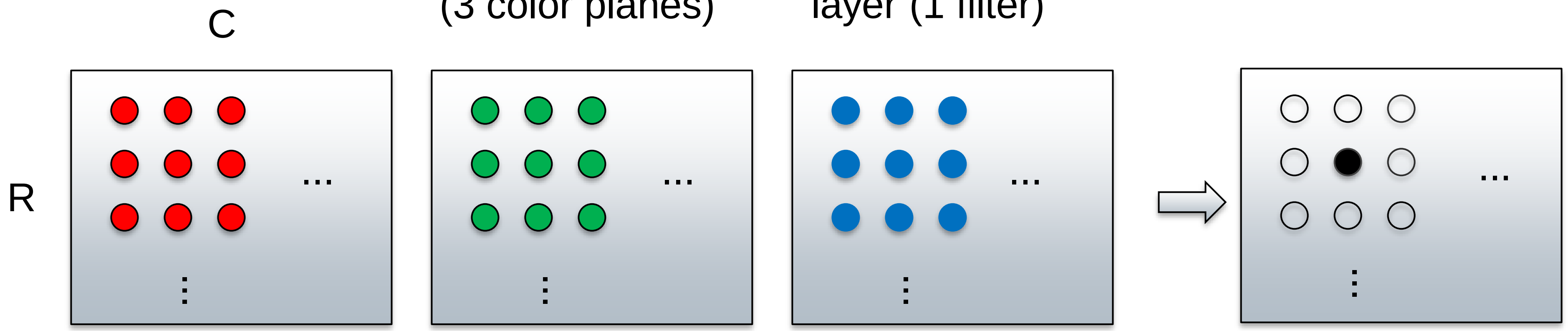
Connections to Filter Plane

- **All the perceptrons in the indicated regions in the RGB planes connect to the center perceptron of the co-located region in the filter plane**



Input layer (3 color planes)

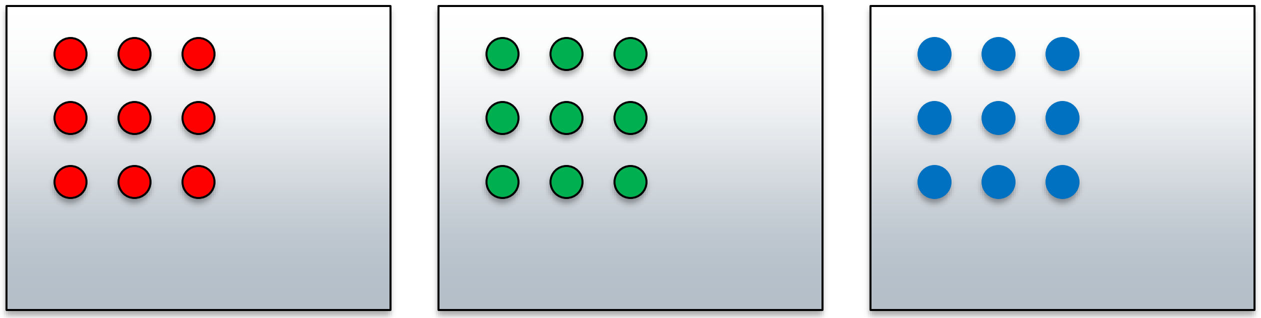
Convolution layer (1 filter)



These 27 pixels connect to.......................this pixel in the filter plane!

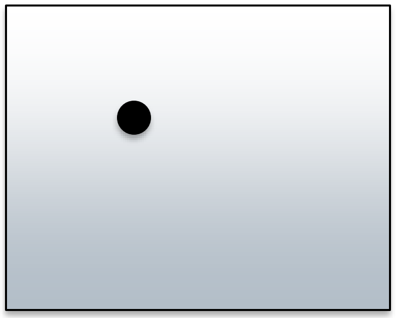
Connections: Another View

- **All the RGB perceptrons below connect to the black filter plane perceptron**



Unstacked input layer planes

One perceptron in one filter plane

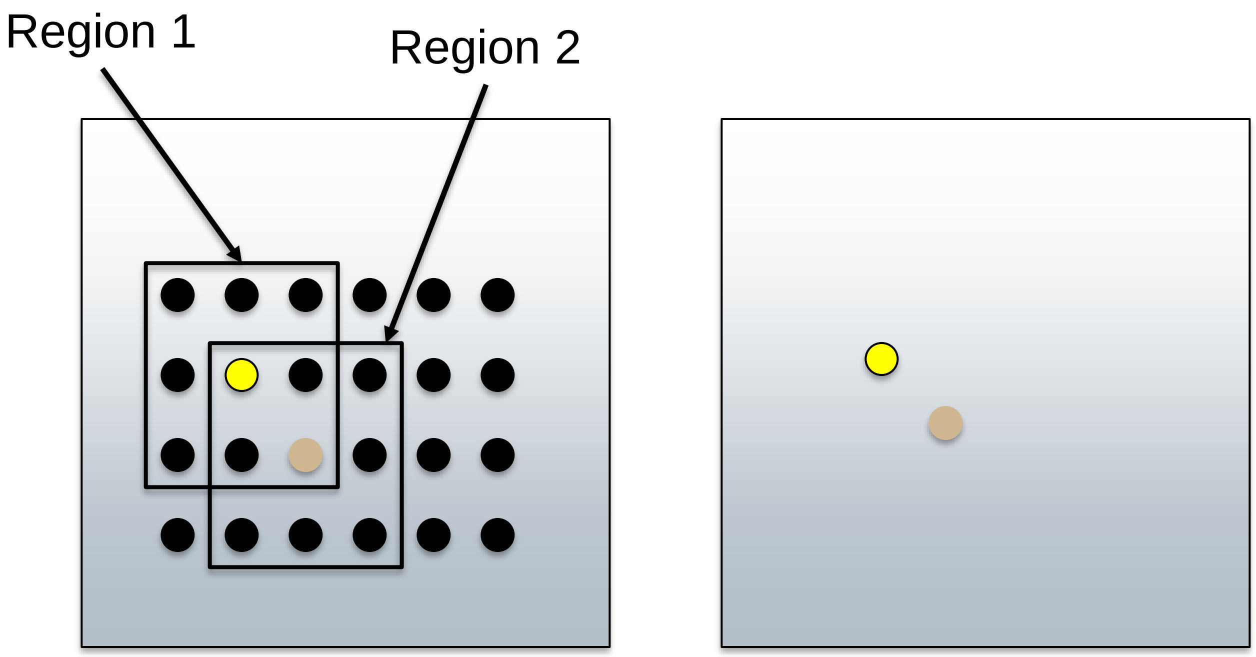


This perceptron is connected to the 27 perceptrons shown in the 3D input volume above

It is the center of the co-located region of the filter plane

Spatial Overlap

- **Filter plane perceptrons connect to overlapping spatial regions in the preceding layer**



preceding layer
depth planes

convolution layer
filter plane

Region 1 perceptrons connect to the yellow perceptron on the right, region 2 perceptrons to the brown one

Since all perceptrons in a filter plane have the same weights, this implements a convolution

Convolution Layer Output Volume

- **Usually the spatial dimensions of a filter plane are the same as the spatial dimensions of the preceding layer**
  - (We're ignoring edge effects. Padding, etc. may be necessary)
- **It is possible to *sub-sample* when building a filter plane, resulting in smaller spatial dimensions**
  - ✓ Example: to downsample by 2 in the previous example, only connect every other set of 3x3 regions in the input layer to a filter plane (can do this in both spatial dimensions)
  - ✓ Caution: this is not recommended practice, it is better to use pooling! (But it will be more computationally expensive)

Pooling Layer

- **Reduces the volume of a preceding layer, thereby reducing the computation of subsequent layers**
  - Helps prevent overfitting since it reduces resolution
- **From a DSP perspective, pooling is downsampling**
  - Factor of 2 is most common
- **MAX pooling is common**
  - Outputs the maximum value among its several inputs
  - Note this layer is *non-parameterized* and therefore unaffected by training
- **The depth dimension is preserved...every depth plane is downsampled independently**

Pooling Layer Illustrated

- **Below illustrates MAX pooling in regions of size 2x2**

12	-4	11	0
8	3	-5	14
9	8	18	6
-1	-2	14	3

Values before MAX pooling

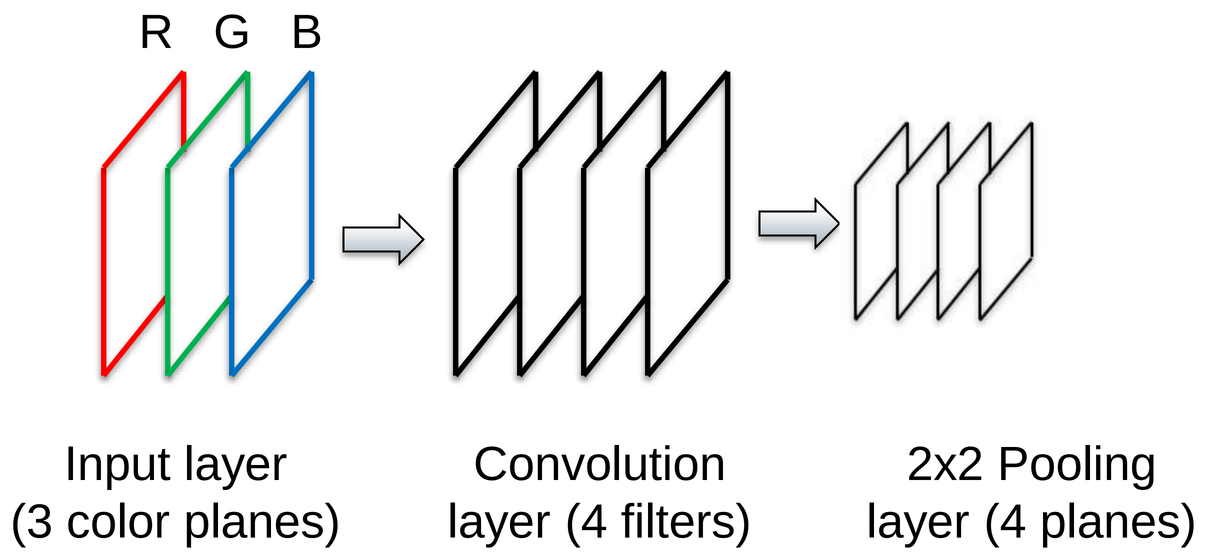
12	14
9	18

Values after 2x2 max pooling

The MAX function looks at its input values and outputs the largest

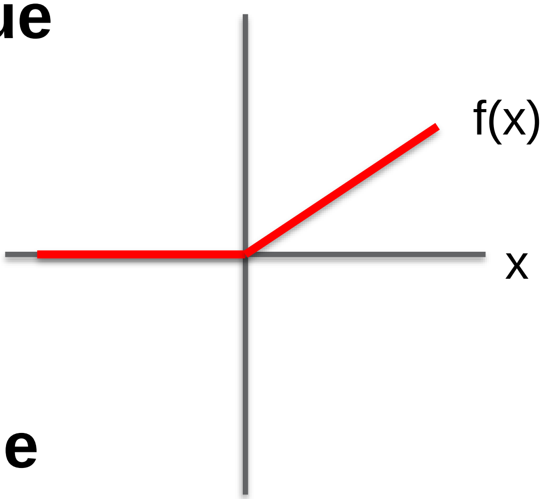
Layer Connections

- **Below shows input / convolution / pooling layers**



RELU Layer

- **Applies a non-linearity to each value**
  - $f(x) = \max(0, x)$
- **Non-parameterized**
- **Each “Rectified Linear Unit” connects to exactly one value in the preceding layer’s output volume**
- **The output volume of a RELU layer equals its input volume**

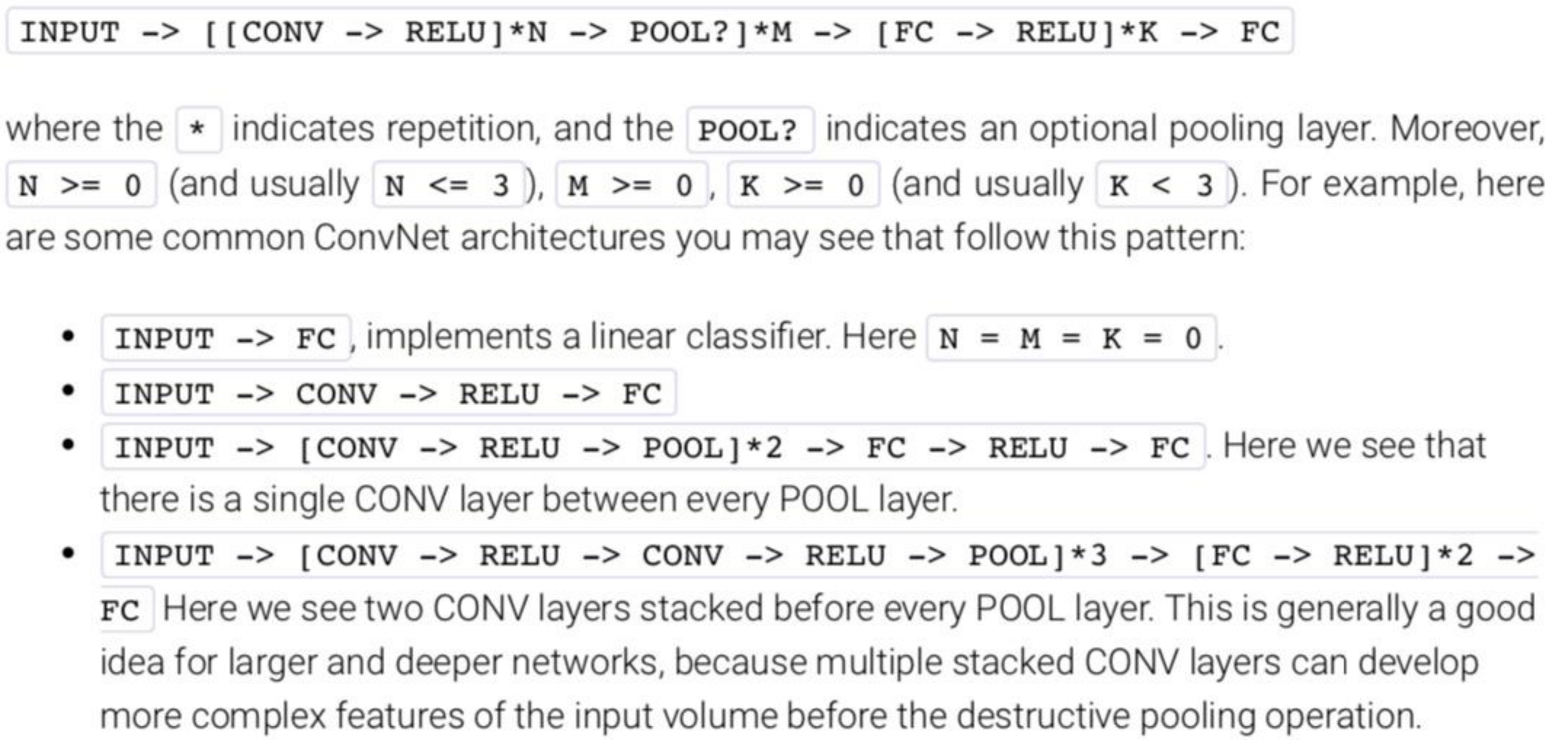


Fully Connected Layer

- **Each perceptron in an FC layer connects to ALL the values in the preceding layer's output volume**
  - This is like ordinary neural networks
- **When classifying**
  - The final layer is FC
  - It has as many perceptrons as there are classes
  - We can visualize this final layer as a 1 x 1 x C output volume where C is the number of classes

Typical Layer Structures

- **From Stanford course notes for CS231n "Convolutional Neural Networks for Visual Recognition"**



`INPUT -> [[CONV -> RELU]*N -> POOL?]*M -> [FC -> RELU]*K -> FC`

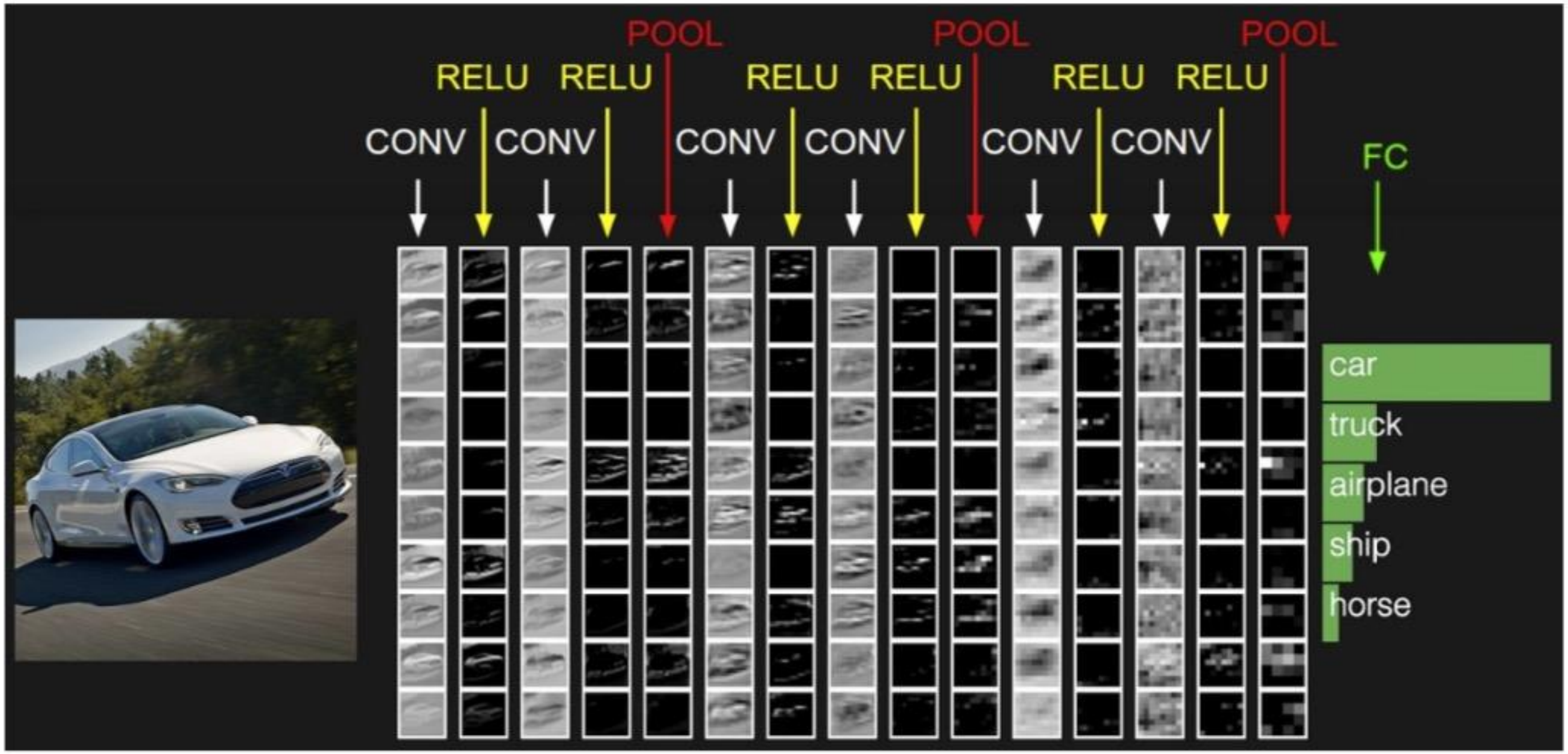
where the `*` indicates repetition, and the `POOL?` indicates an optional pooling layer. Moreover, `N >= 0` (and usually `N <= 3`), `M >= 0`, `K >= 0` (and usually `K < 3`). For example, here are some common ConvNet architectures you may see that follow this pattern:

- `INPUT -> FC`, implements a linear classifier. Here `N = M = K = 0`.
- `INPUT -> CONV -> RELU -> FC`
- `INPUT -> [CONV -> RELU -> POOL]*2 -> FC -> RELU -> FC`. Here we see that there is a single CONV layer between every POOL layer.
- `INPUT -> [CONV -> RELU -> CONV -> RELU -> POOL]*3 -> [FC -> RELU]*2 -> FC` Here we see two CONV layers stacked before every POOL layer. This is generally a good idea for larger and deeper networks, because multiple stacked CONV layers can develop more complex features of the input volume before the destructive pooling operation.

http://cs231n.github.io/convolutional-networks/

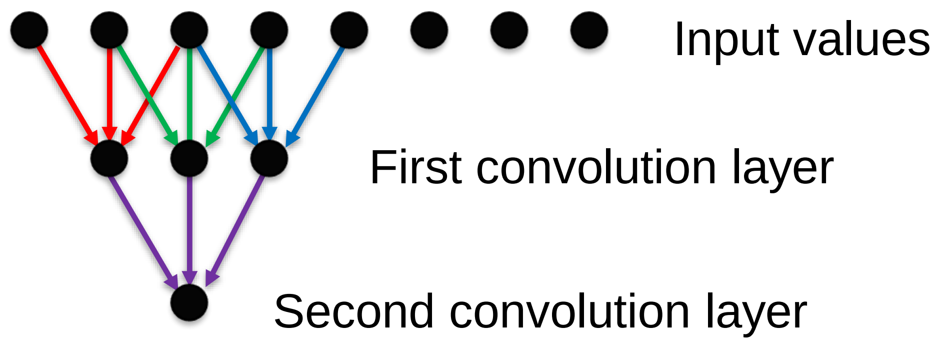
Example Layer Structures

- For recognizing vehicle types



Why Repeat CONV->RELU layers?

- **Allows a larger extent of the image to be analyzed**
  - Assume first CONV layer is 3x3
  - A second 3x3 CONV layer processes the first CONV layer's output and operates on a region of size 5



In this 1-D example, the second convolution layer is processing data over a width of 5 input values

Some Rules of Thumb

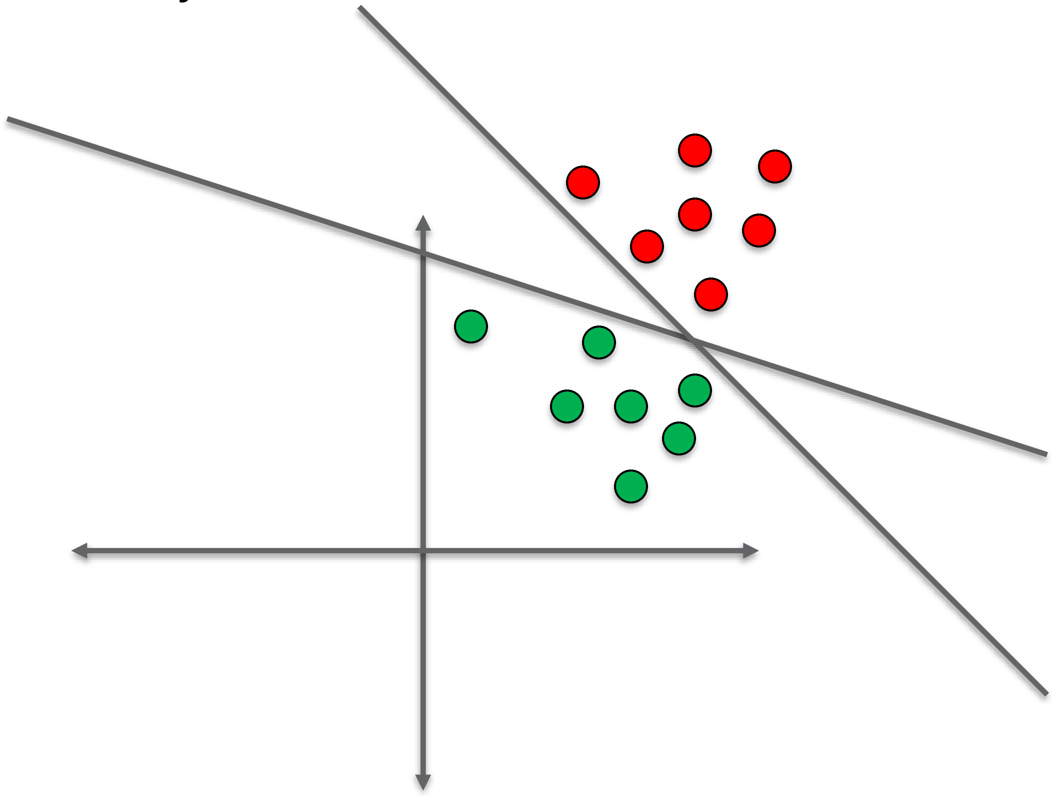
- **Input layer should be divisible by 2 many times**
- **Use 3x3 or 5x5 filters in the CONV layers and do not use them to sub-sample (use POOL layers for this)**
  - Zero-pad at image edges so that CONV layer spatial extent equals that of its input layer
- **Use MAX for the POOL function in a 2x2 region as shown previously**
  - People have experimented with averaging, for example, but MAX works better

SVM’s Conceptually

What if there are many separating hyperplanes?

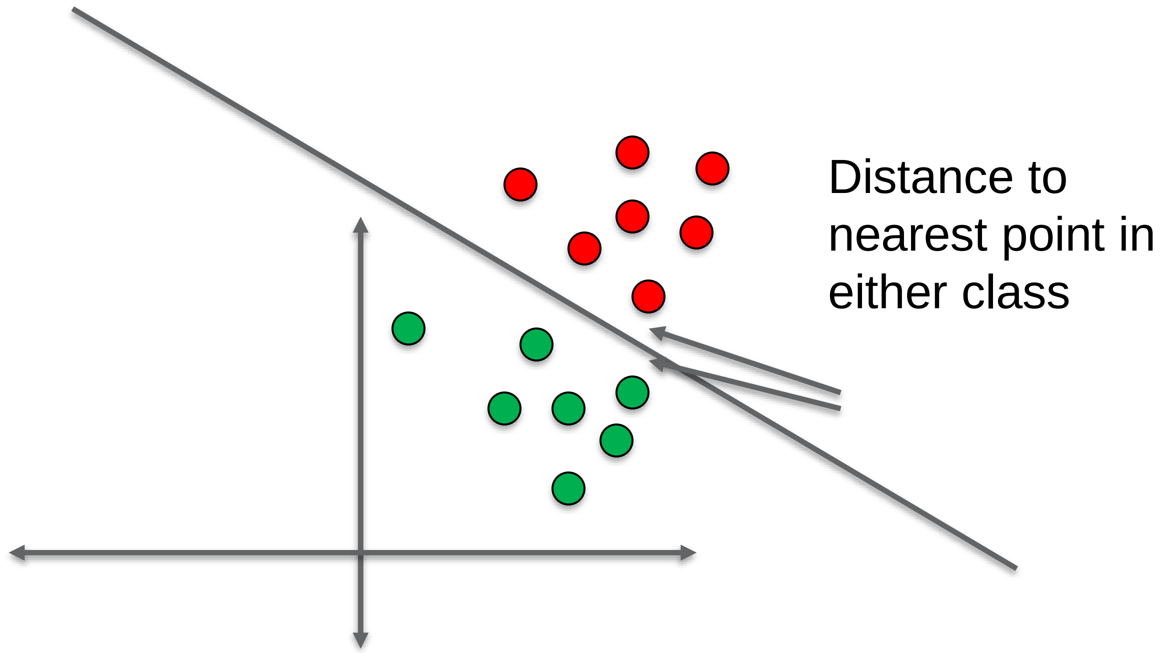
- **When the separating hyperplane is not unique, which one should we choose?**

Both these hyperplanes separate the data, as do many others



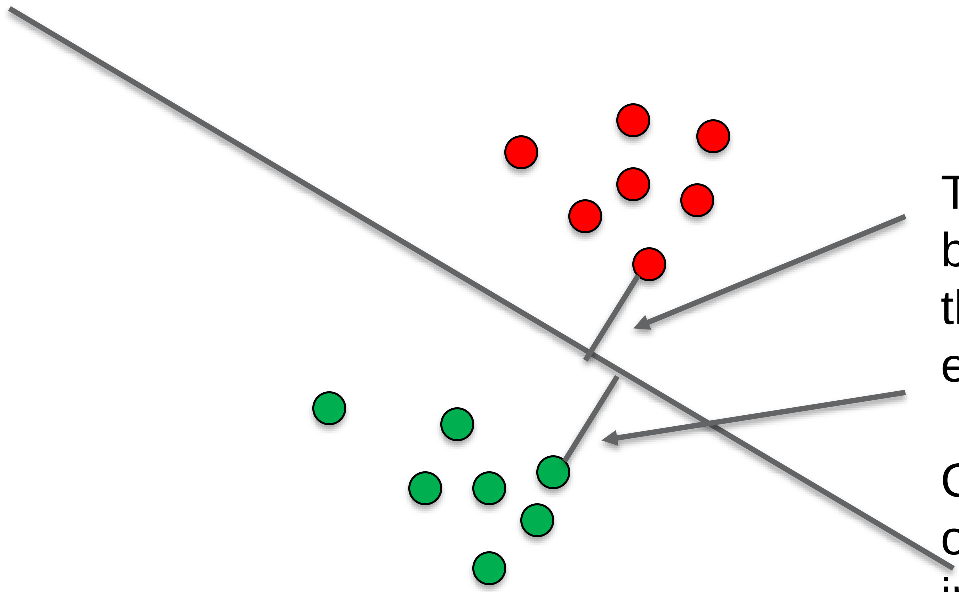
Solution: pick the one that *maximizes the margin*

- **Choose the hyperplane that maximizes the minimum distance to any point in either set**
  - Twice this minimum distance is called the "margin"



The SVM hyperplane

- **A zoomed in picture of the previous slide**

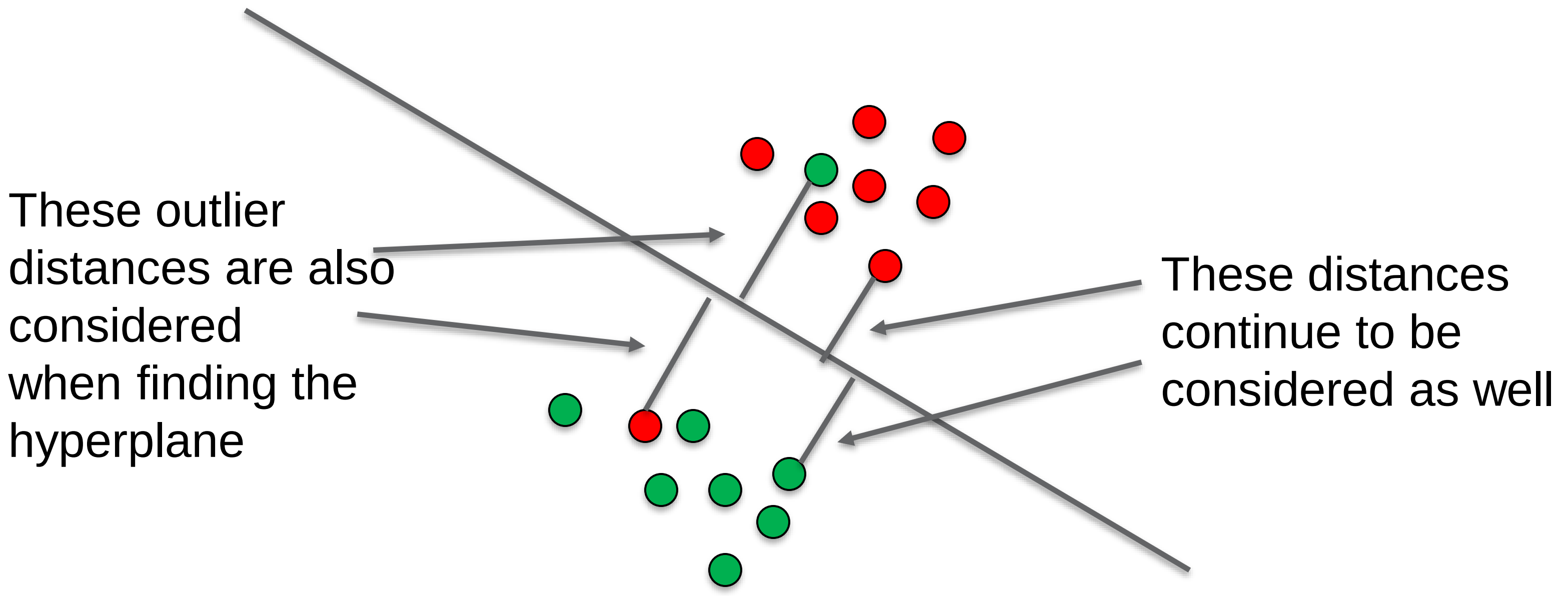


The best hyperplane will be equally distant from the nearest point in each cluster.

Otherwise, moving it one way or the other will increase the minimum distance!

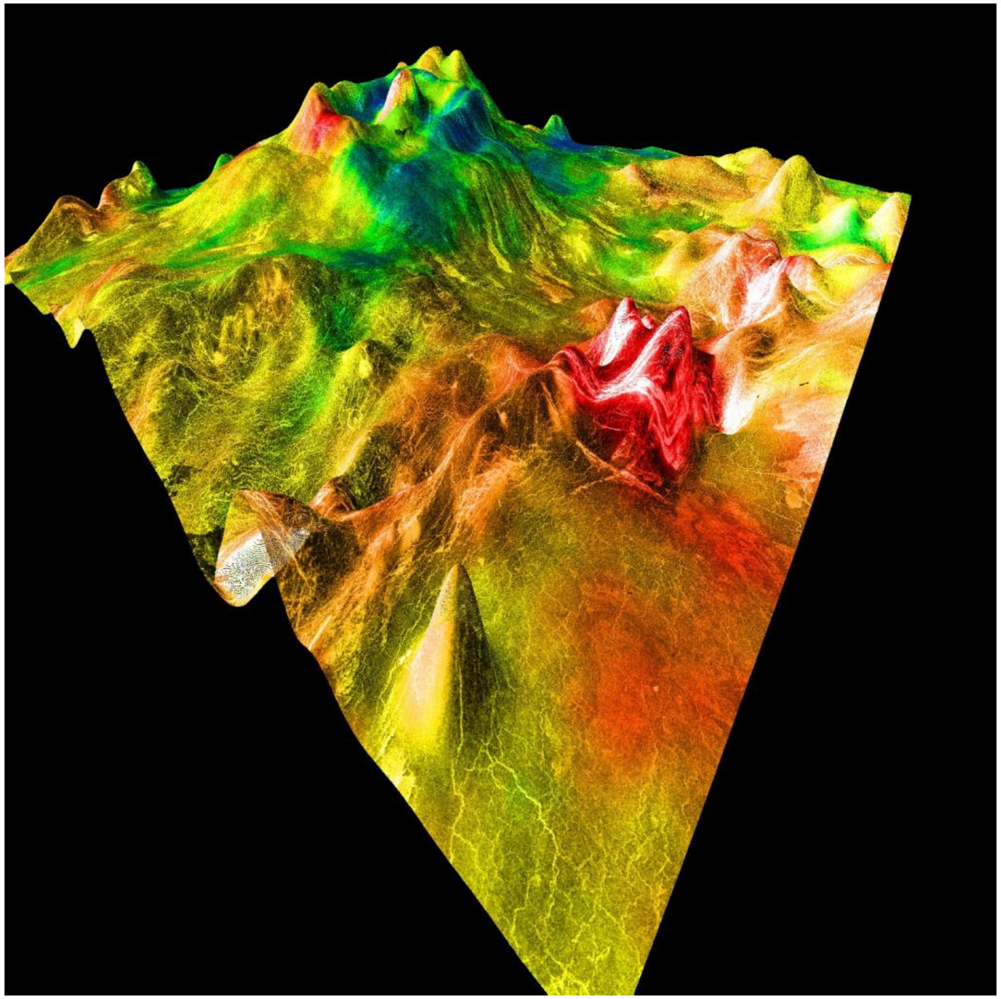
What if the data isn't linearly separable?

- **A different more complicated optimization can be performed that takes into account outliers**


These outlier distances are also considered when finding the hyperplane
These distances continue to be considered as well

Gradient Descent Optimization

A complex 3-D surface



Venus volcano (Smithsonian Institution image)

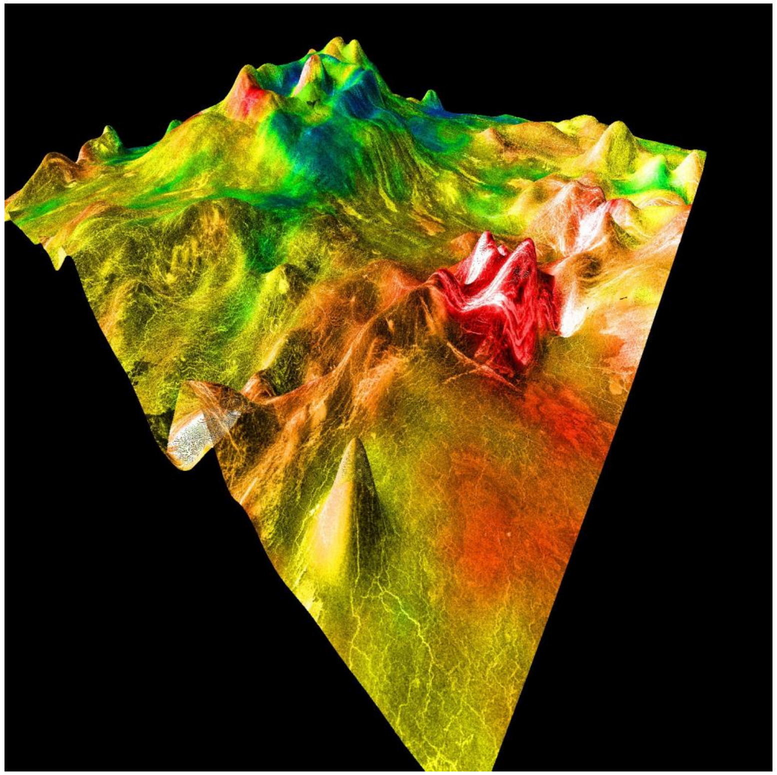
Imagine starting at the highest point in this picture.

You can only see the area in your immediate vicinity.

How can you find your way down as fast as possible?

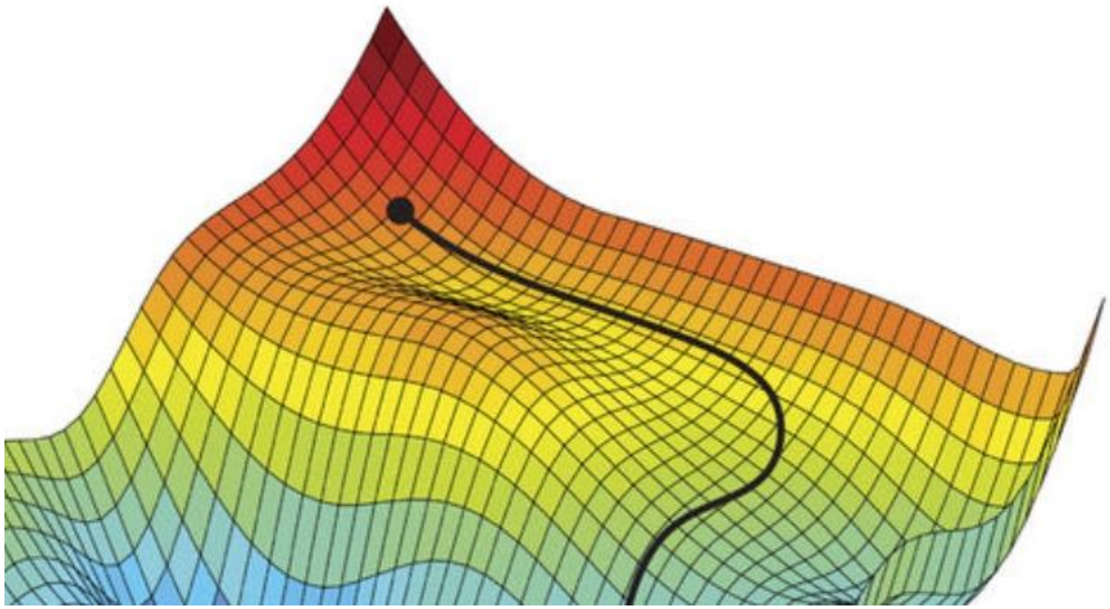
A complex 3-D surface

- **What path would a stream of water follow?**
  - At each point, it would move downhill in the steepest direction



Water flowing illustration

- **The water finds its way downhill quickly...at least we hope!**

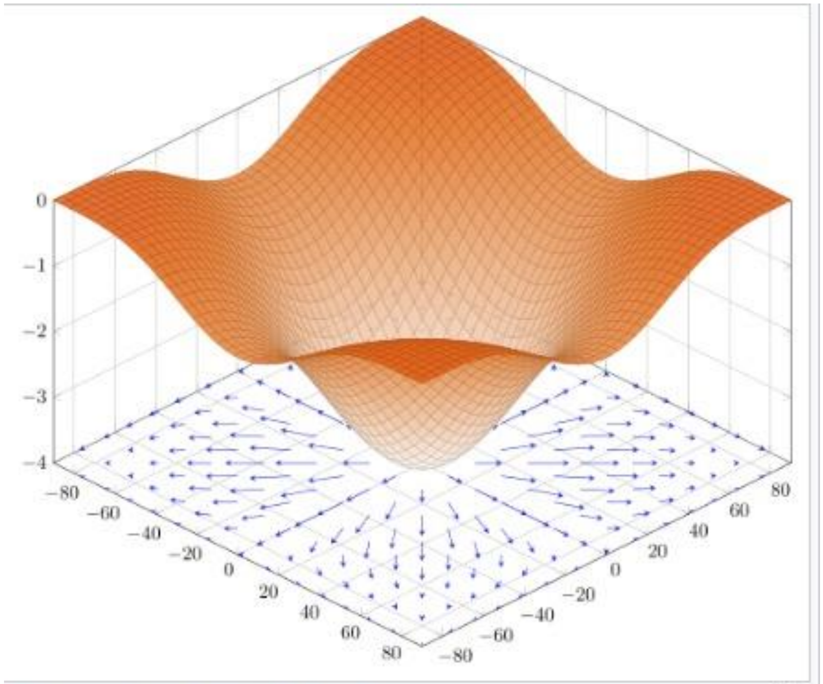


https://www.sciencemag.org/news/2018/05/ai-researchers-allege-machine-learning-alchemy

The Gradient

Carnegie Mellon Africa

- **The vector pointing in the direction of steepest *increase* at each point is the gradient**
  - The *negative* of the gradient points in the direction of steepest descent



The gradient of the function $f(x,y) = -(\cos^2 x + \cos^2 y)^2$ depicted as a projected vector field on the bottom plane.

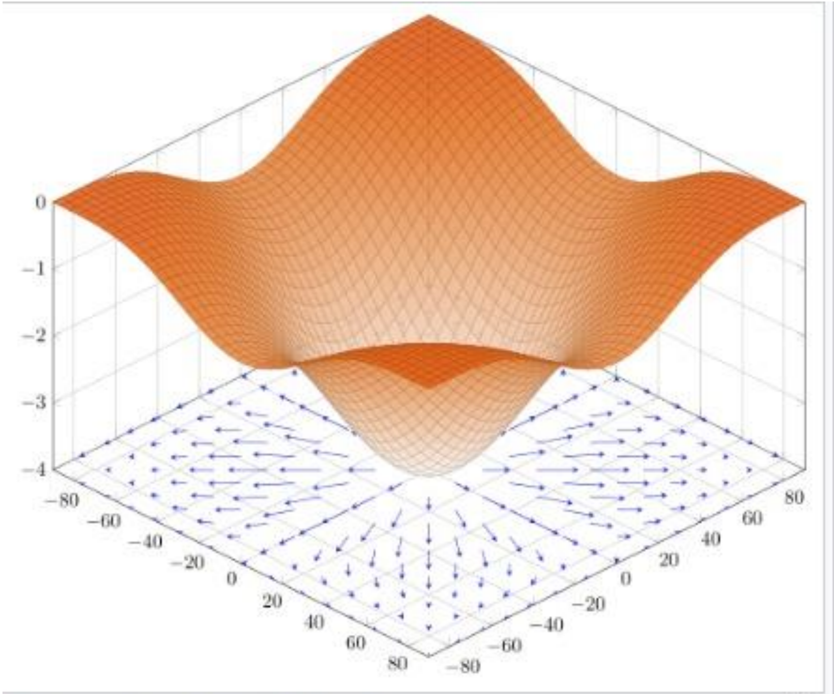
wikipedia

The Gradient Vector

Carnegie Mellon Africa

- **The gradient is computed using partial derivatives**
- **The gradient of *f(x,y)* is given by**

$$\nabla_f = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right)$$

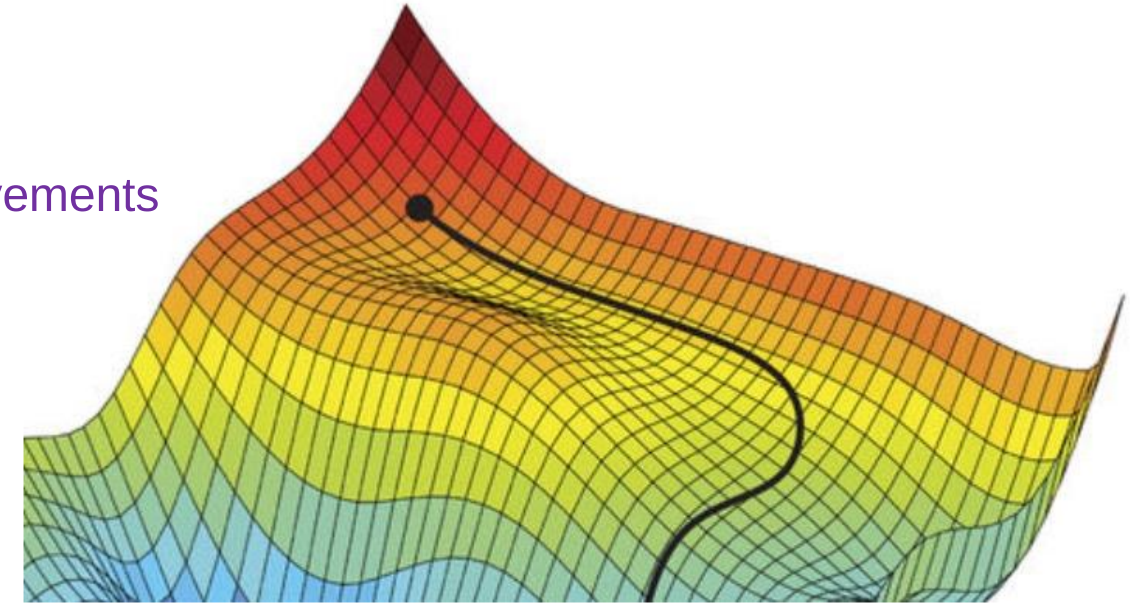


The gradient of the function $f(x,y) = -(\cos^2 x + \cos^2 y)^2$ depicted as a projected vector field on the bottom plane.

The Descent Algorithm

- **At each point**
  - Compute the gradient vector
  - Move a short distance in the *opposite* direction
  - Repeat!

Stop iterating when improvements become small



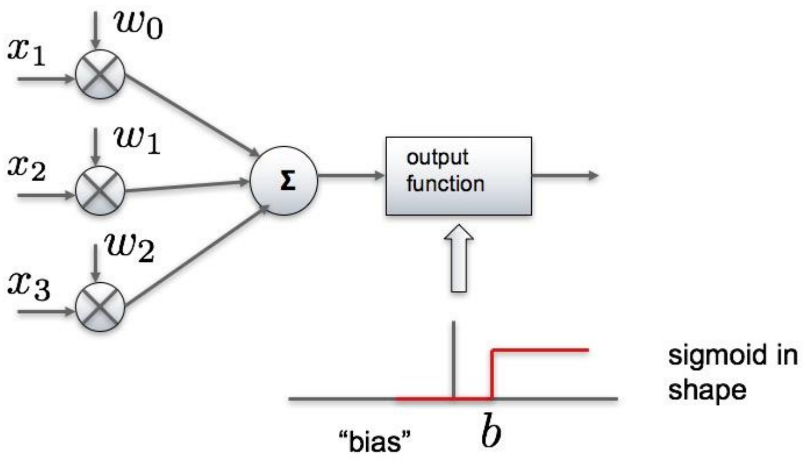
https://www.sciencemag.org/news/2018/05/ai-researchers-allege-machine-learning-alchemy

Neural network gradient descent

- **The function being minimzed is the total cost**

$$C(TS)$$

- **The variables are the percpetron weights and biases**



The Back propagation Algorithm

- **The partial derivatives are computed using the *back propagation* algorithm**
  - Details are covered in other classes at CMU
- **The distance moved each step controls how fast the network learns**
  - Moving too far each step may cause oscillation or prevent convergence
- **Each gradient descent step is called a *model update***

Stochastic Gradient Descent

- **Complication**
  - If the entire TS is used to estimate the gradient before taking a downhill step, convergence can be very slow
- **Solution**
  - Estimate the gradient using just a few instances from the TS before updating the model
    - ✓ This is called *mini-batch gradient descent*
  - Sometimes the mini-batch size is set to one
    - ✓ This is called *stochastic gradient descent*

Note that both mini-batch gradient descent and stochastic gradient descent only estimate the true gradient before updating the model parameters