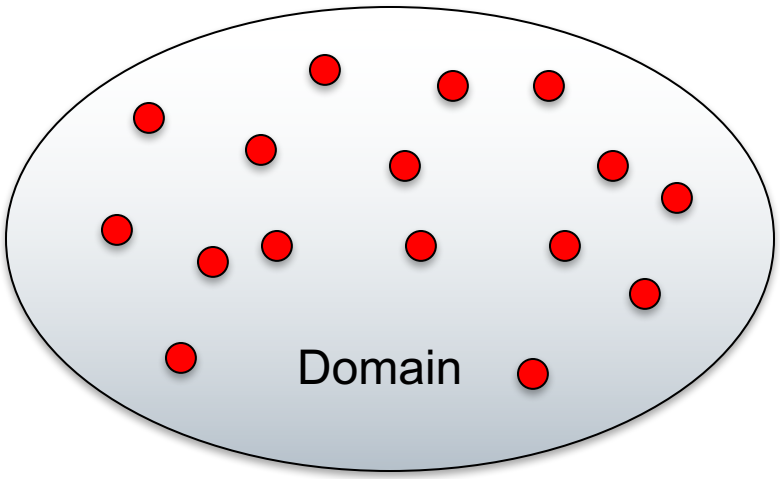
Introduction to Neural Networks

Machine Learning: Classification

- **Consider a domain of objects that we wish to classify**
  - Example: the domain could be passengers on the Titanic
    - ✓We want to classify them as “survived” or “perished”



The red dots represent objects in the domain

Example: the set of passengers on the titanic

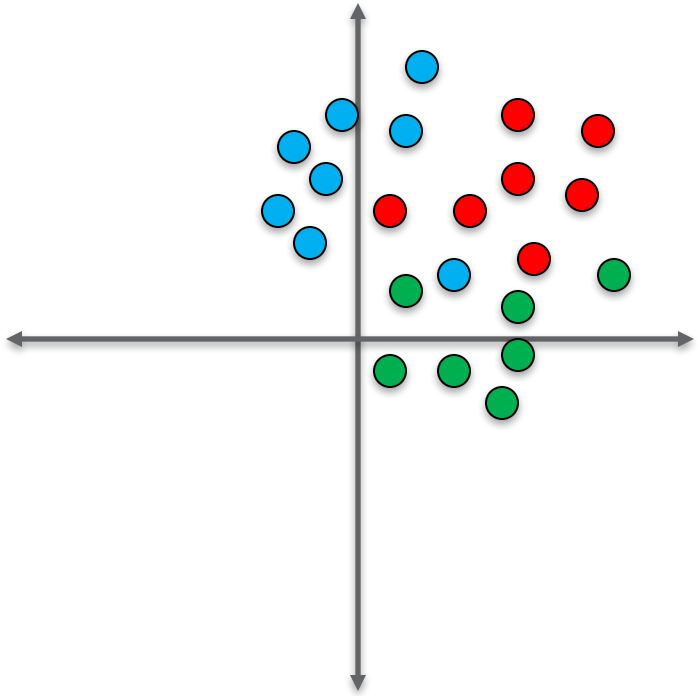
Extracting Features from the Objects

- **The objects to classify are often too complex to process directly**
  - Example: how do you represent everything it means to be a unique human individual?
- **Solution:**
  - Process the objects to extract a set of features
  - Group the features together into a feature vector
  - We refer to a feature vector as an instance
  - We refer to the set of all feature vectors as the instance space

Example Titanic feature vector: (Age, Gender, Fare class, Cabin deck)

Visualizing Instances

- **Assume the feature vector (instance) is 2D and there are 3 classes in our problem**
  - 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)

The Training Set

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

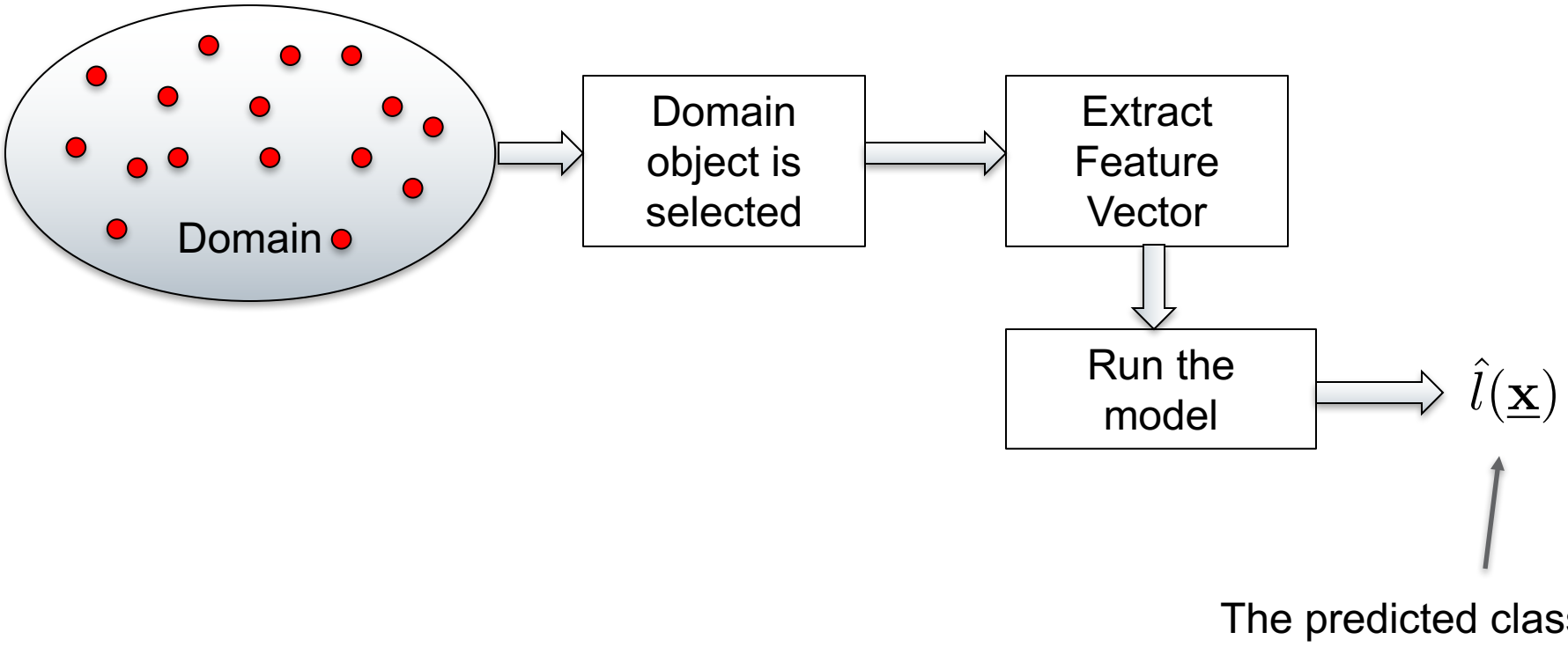
$$\mathcal{T} = \{\underline{\mathbf{x}}_i, l(\underline{\mathbf{x}}_i)\}$$

- **Here $\underline{x}_i$ is an instance and *l( )* is the true labelling function (i.e. its value is the correct class)**
- **The training set is samples of the true classification function**
  - It doesn't not contain all possible feature vectors
- **The function we learn should approximate the true function**

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

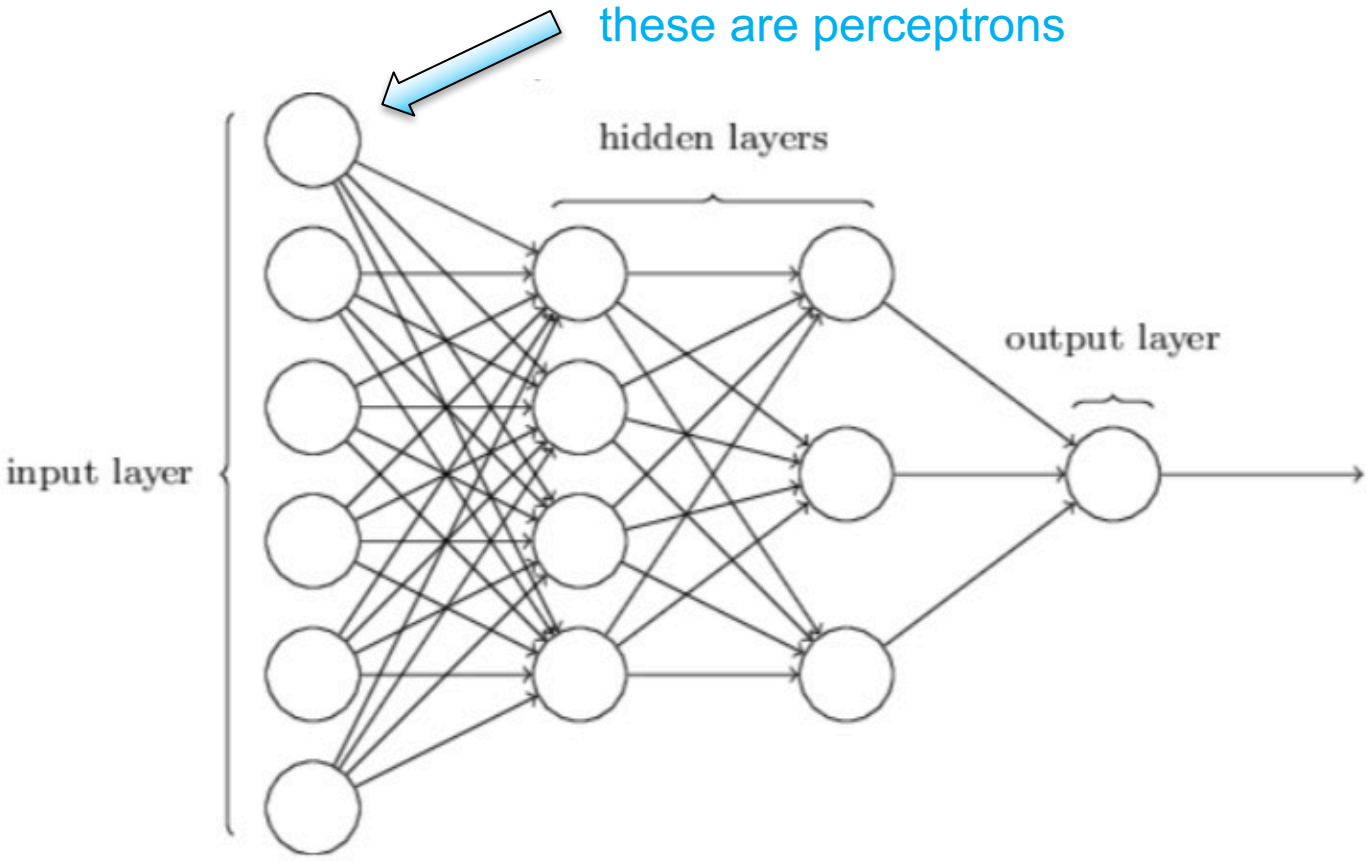
Diagram of system's use after training

- **An object from the domain is "selected" by the problem at hand**
  - Example: an image arrives from a security camera



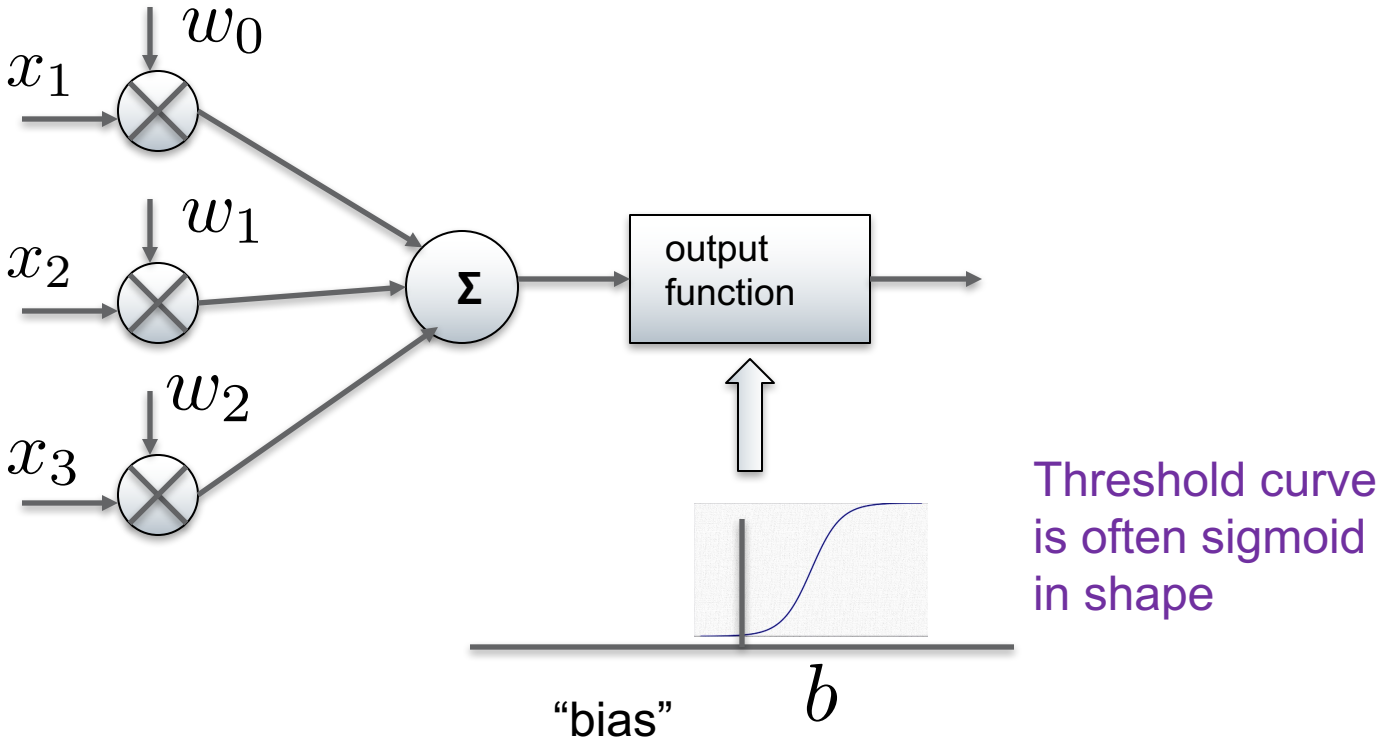
Neural Networks Diagram

- **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***



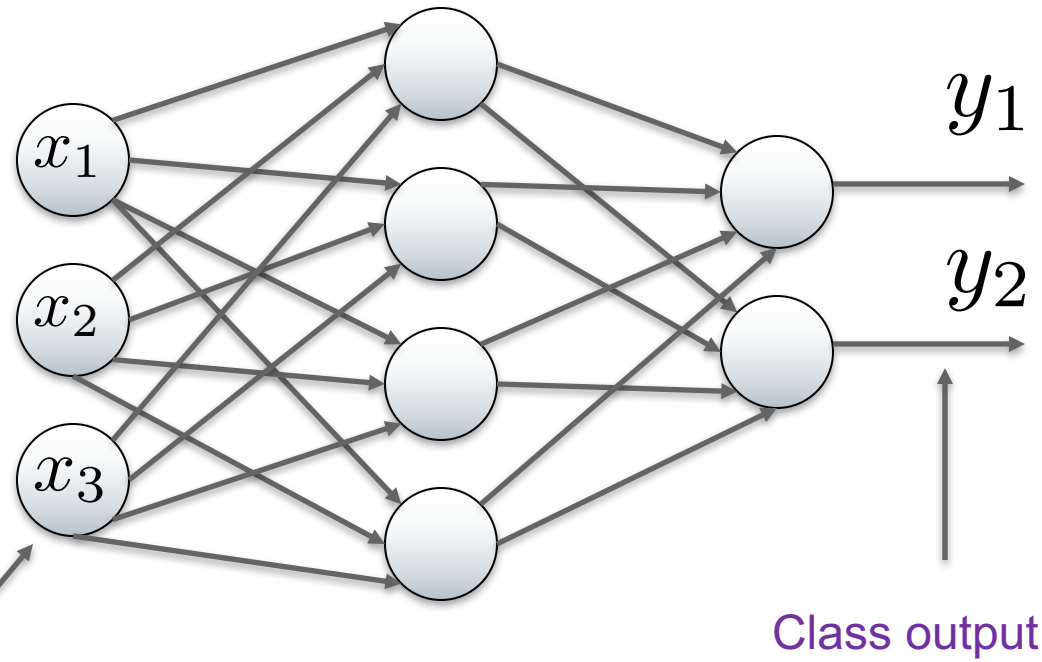
Threshold curve is often sigmoid in shape

Neural Networks for Classification

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

Neural Network Classification

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

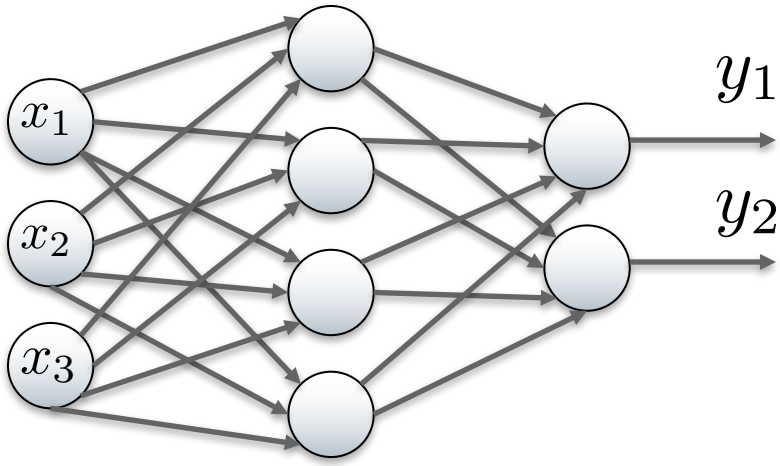


Neural network training overview

- **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 used to find good weights and biases**
  - We are searching through a high-dimensional space for a solution
- **The back-propagation algorithm is used to compute the gradient**

Cost function for optimization

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



Let the networks output when $\underline{\boldsymbol{x}}$ is applied by $\underline{\boldsymbol{y}}(\underline{\boldsymbol{x}})$

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

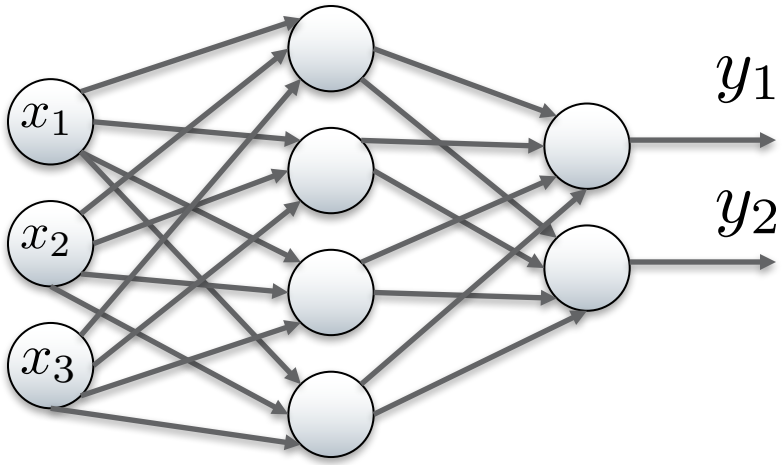
The cost function for $\underline{\boldsymbol{x}}$ is

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

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

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

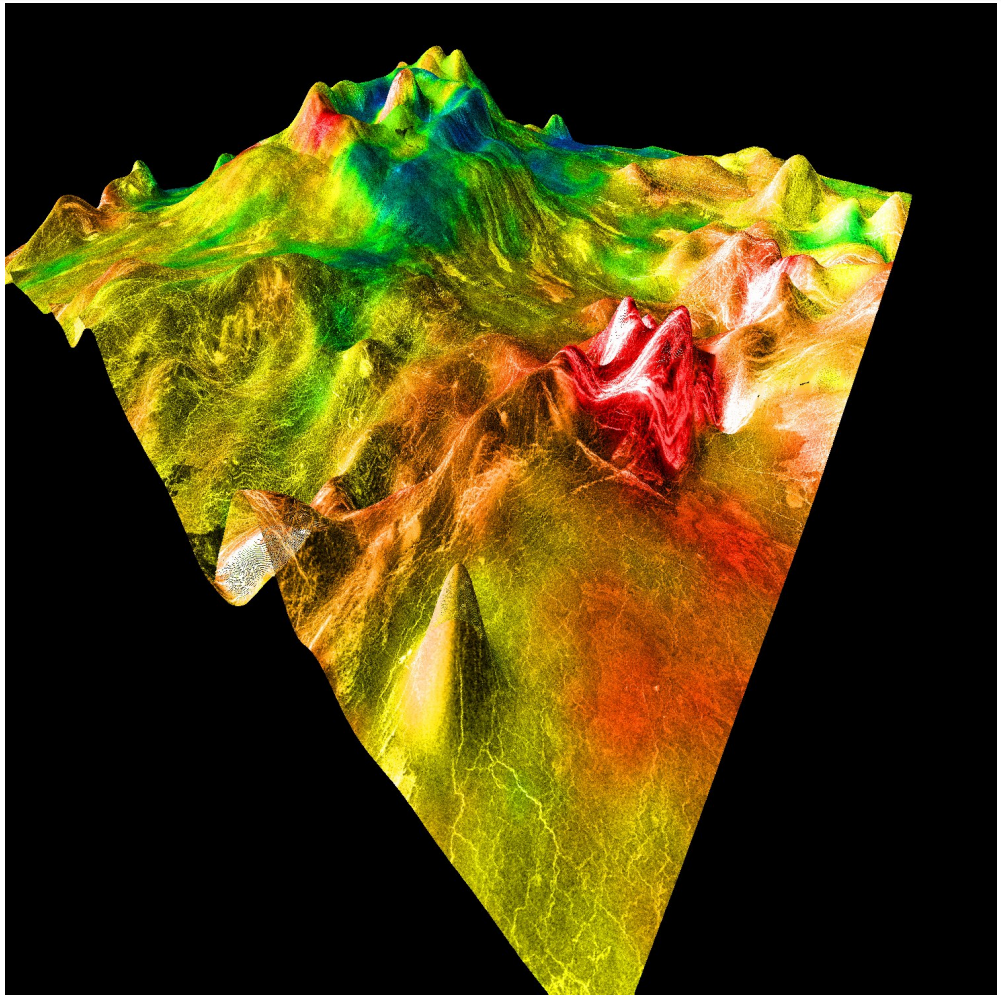


$$C(\mathrm{TS}) = \sum_{\underline{\mathbf{x}} \in \mathrm{TS}} \|\underline{\mathbf{l}}(\underline{\mathbf{x}}) - \underline{\mathbf{y}}(\underline{\mathbf{x}})\|^2$$

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

Optimization with Gradient Descent

A complex 3-D surface



Venus volcano (Smithsonian Institution image)

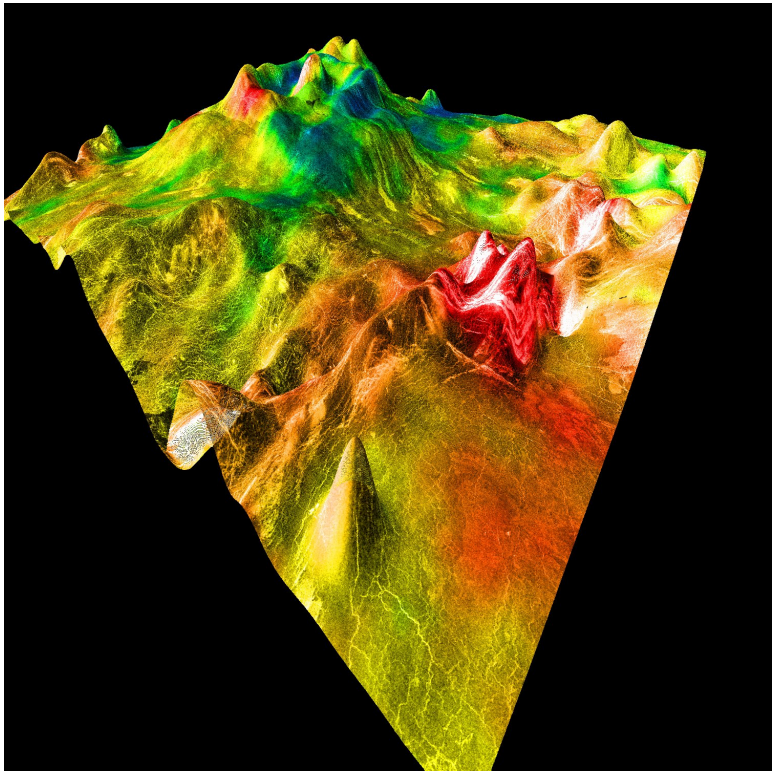
Imagine starting at the highest point in this picture.

You can only see the area in your immediate vicinity.

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

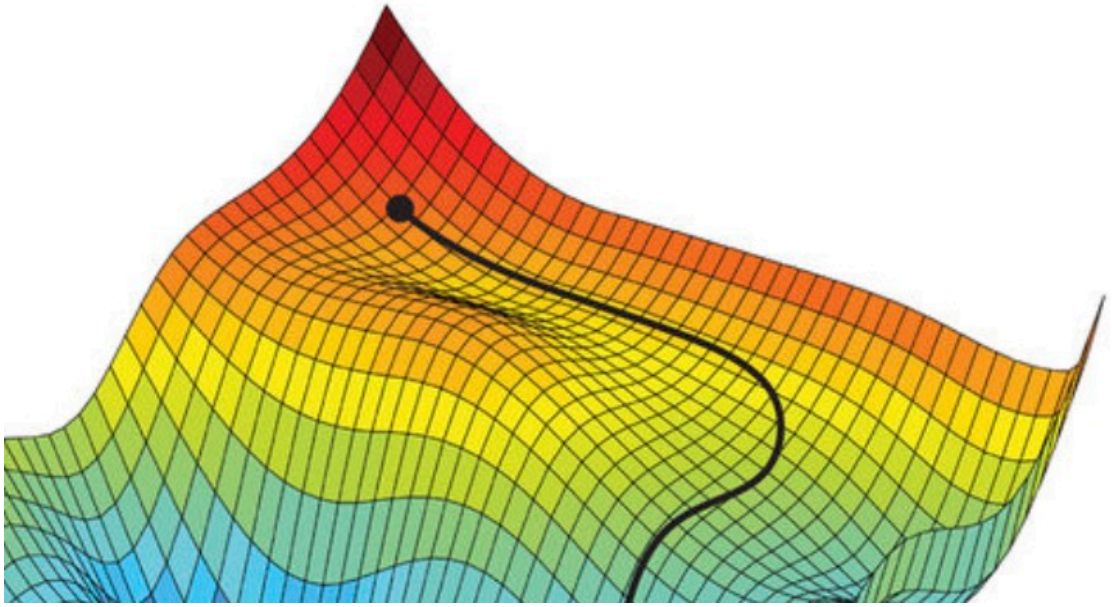
A complex 3-D surface

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



Water flowing illustration

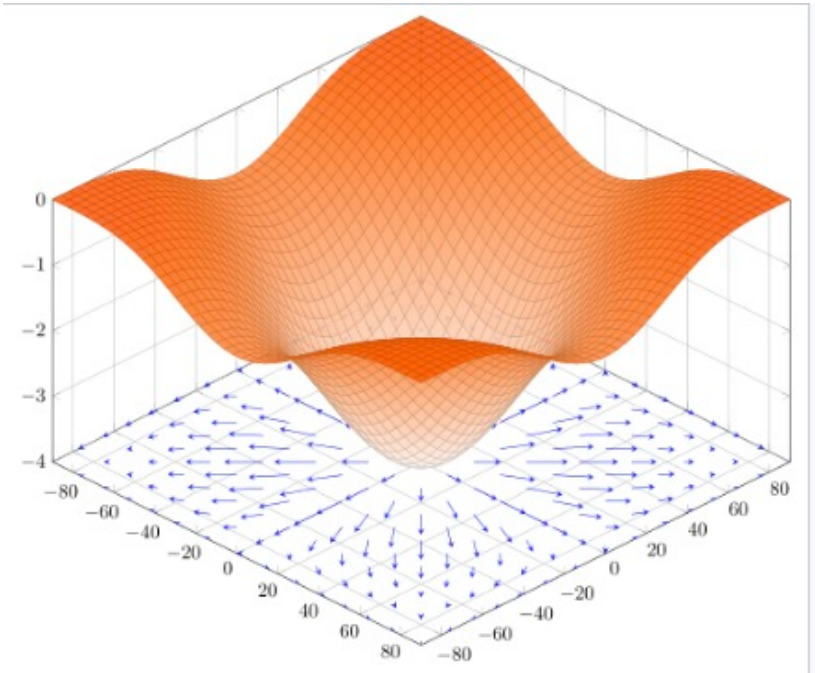
- **The water finds its way downhill quickly...at least we hope!**



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

The Gradient

- **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



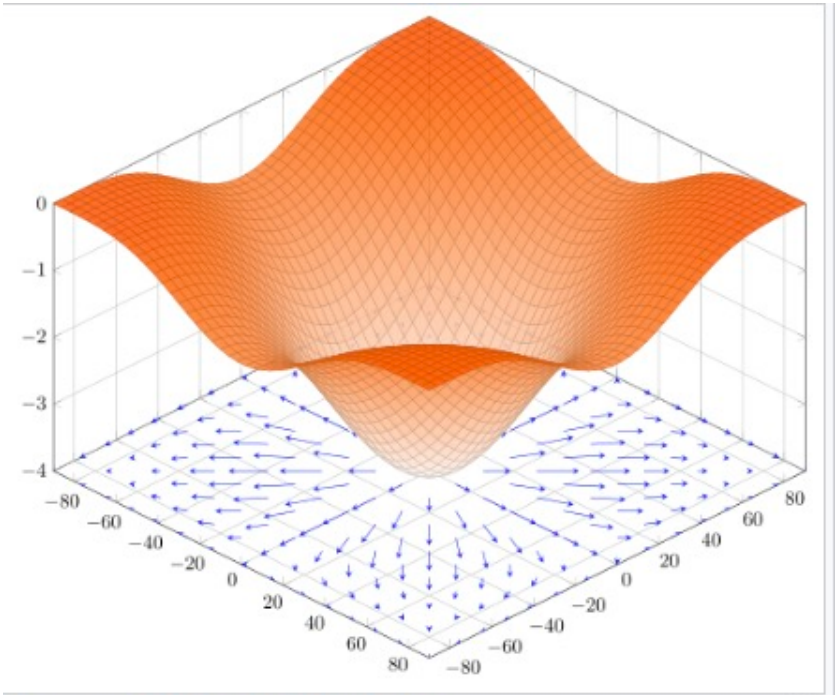
The gradient of the function $f(x,y) = -(\cos^2 x + \cos^2 y)^2$ depicted as a projected vector field on the bottom plane.

wikipedia

The Gradient Vector

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

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

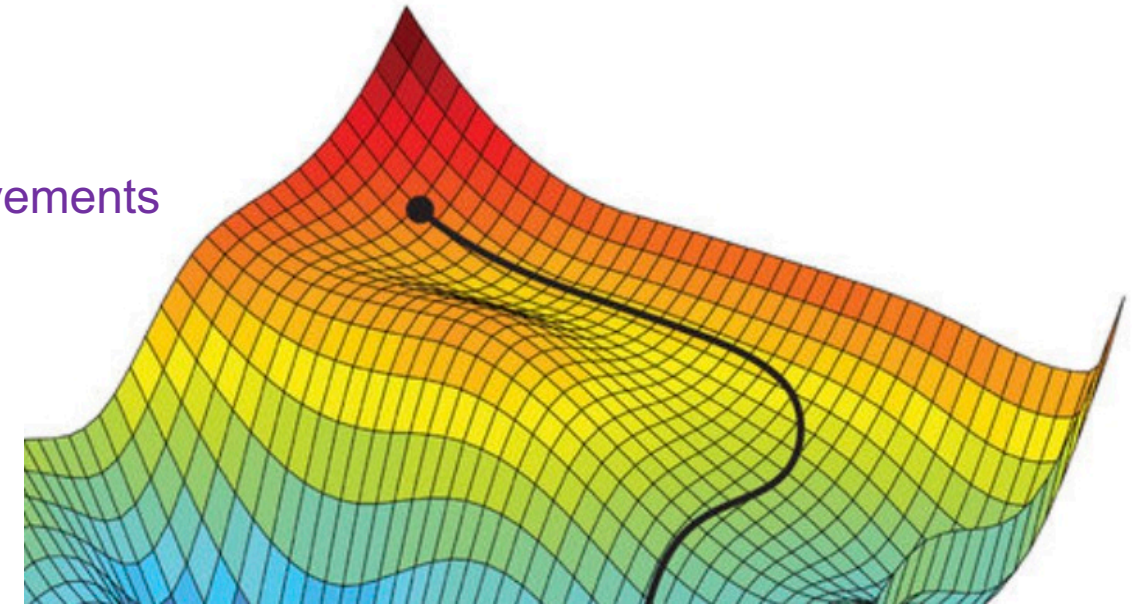


The gradient of the function $f(x,y) = -(\cos^2 x + \cos^2 y)^2$ depicted as a projected vector field on the bottom plane.

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



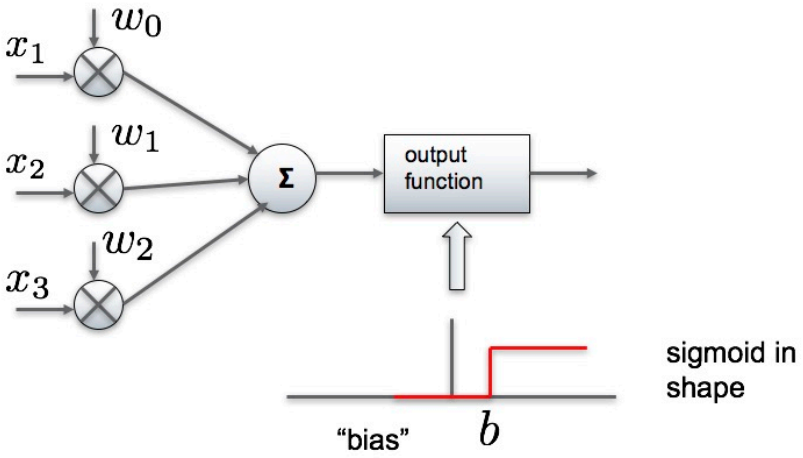
https://www.sciencemag.org/news/2018/05/ai-researchers-allege-machine-learning-alchemy

Neural network gradient descent

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

$$C(\mathrm{TS}) = \sum_{\underline{\mathbf{x}} \in \mathrm{TS}} \|\underline{\mathbf{l}}(\underline{\mathbf{x}}) - \underline{\mathbf{y}}(\underline{\mathbf{x}})\|^2$$

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



The Back propagation Algorithm

- **The partial derivatives are computed using the "back propagation" algorithm**
  - For details take a class on ML
- **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
    - ✓ This is called *batch gradient descent*
- **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 gradient before updating the model parameters