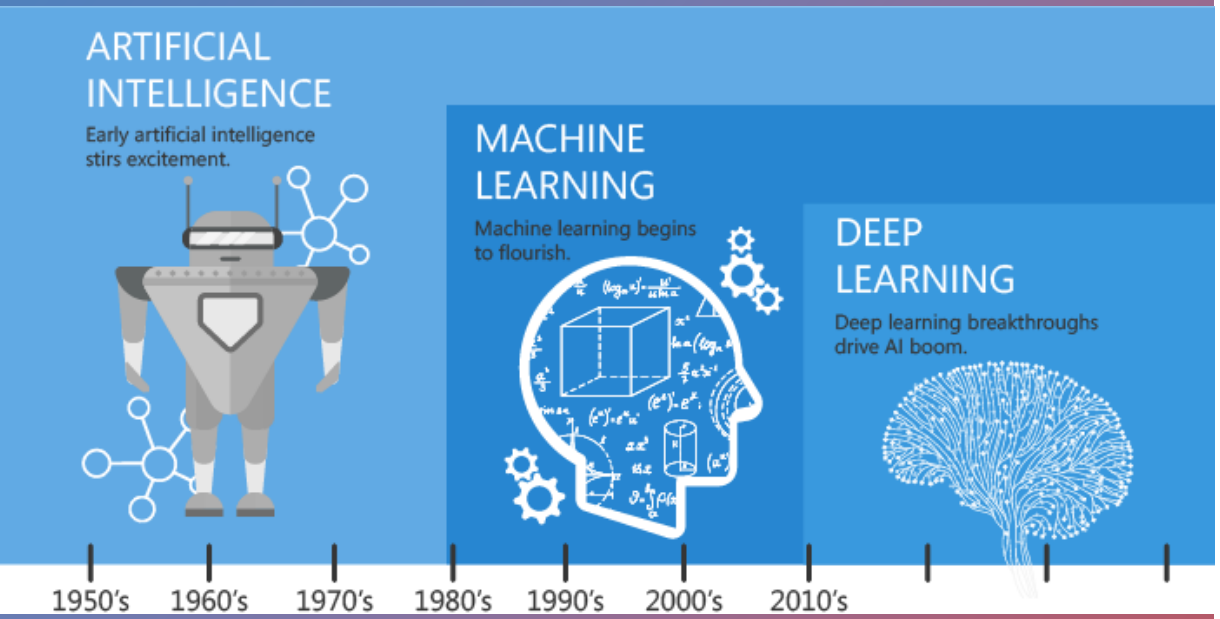


CSPE72 – Deep Learning Techniques

Outline

- Introduction
- Machine learning
- Deep learning Techniques

Course Objectives

- To introduce building blocks of deep neural network architecture
- To learn deep learning algorithms and its problem settings
- To understand representation and transfer of knowledge using deep learning
- To learn to use deep learning tools and framework for solving real-life problems
- To use Python for Deep Learning

Content

UNIT I: Deep Networks Deep Feedforward Networks - Learning XOR - Gradient Based learning - Hidden Units - Back-propagation and other Differential Algorithms - Regularization for Deep Learning - Optimization for training Deep Models.

UNIT II: Convolutional Networks Convolution operation - Motivation - Pooling - Convolution and Pooling as strong prior - Efficient convolution algorithms - Unsupervised features - Sequence Modeling: Recurrent and Recursive Nets - LSTM Networks - Applications - Computer Vision - Speech Recognition - Natural Language Processing.

UNIT III: Linear factor Models Probabilistic PCA and Factor Analysis - Independent Component Analysis (ICA) - Auto encoders - Regularized Auto encoders - Representational Power - Layer size and Depth - Stochastic Auto encoders - Applications.

UNIT IV: Representation Learning Greedy Layer-wise Unsupervised Pre-Training - Transfer learning and Domain Adaptation - Deep Generative Models.

UNIT V: Deep Learning with Python Introduction to Keras and Tensorflow - Deep Learning for computer vision - convnets - Deep Learning for Text and Sequences - Generative Deep Learning - Text Generation with LSTM - DeepDream - Neural Style Transfer - Generating images with variational autoencoders - Generative Adversarial Networks (GAN).

Books

Text Book

1. Ian Goodfellow, Yoshua Bengio, Aaron Courville, “Deep Learning”, The MIT Press, 2016.

Reference Books

1. Francois Chollet, “Deep Learning with Python”, Manning Publications, 2017.
2. Aurélien Géron, “Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems”, First Edition, O'Reilly Media, 2017.
3. Josh Patterson, “Deep Learning: A Practitioner's Approach”, First Edition, O'Reilly Media.

UNIT I

Deep Networks: Deep Feedforward Networks - Learning XOR - Gradient Based learning - Hidden Units - Back-propagation and other Differential Algorithms - Regularization for Deep Learning - Optimization for training Deep Models.

Introduction

Machine Learning

- Machine learning, a branch of artificial intelligence, concerns the construction and study of systems that can learn from data.
- The phrase "machine learning" describes a wide diversity of algorithms and techniques.

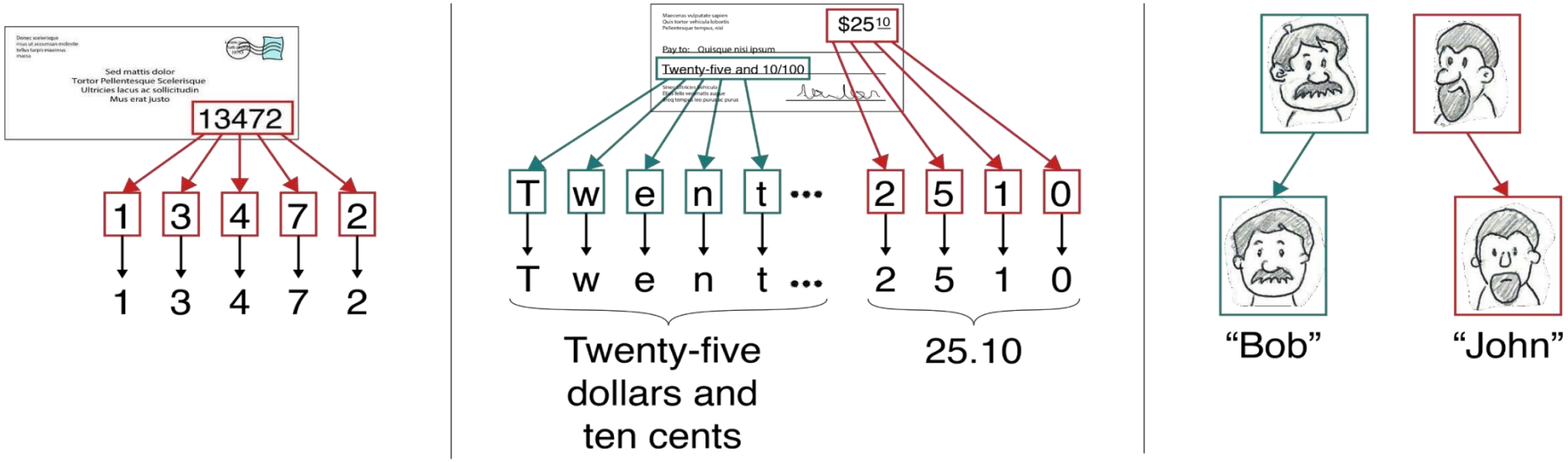
A big, expanding collection of algorithms and principles that analyze vast quantities of training data in order to extract meaning from it.

- The phrase machine learning describes a growing body of techniques that all have one goal: discover meaningful information from data.

Data

- Data can be
  - raw numbers (like stock prices on successive days, the mass of different planets, the heights of people visiting a county fair).
  - sounds (the words someone speaks into their cell phone),
  - pictures (photographs of flowers or cats),
  - words (the text of a newspaper article or a novel),
  - or anything else that we want to investigate.
- "Meaningful information" is whatever we can extract from the data that will be useful to us in some way.
- We decide what's meaningful to us, and then we design an algorithm to find as much of it as possible from our data.

Example applications that use machine learning to extract meaning from data



Example applications that use machine learning to extract meaning from data

- **Left: Getting a zip code from an envelope.**
- **Middle: Reading numbers &letters on a cheque.**
- **Right: Recognizing faces from photos.**

Expert systems can also find meaning from data

- Expert systems was an early approach to find the meaning that's hiding inside of data.
- Idea:
  - Study what human experts know and automate that.
  - Make a computer mimic the human experts it was based on.
- Create a rule-based system: a large number of rules for the computer to imitate human experts.
- Example: Recognize zip codes. 7's have a horizontal line at the top, and a diagonal line that starts at the right edge of the horizontal line and moves left and down. Some people put a bar through the middle of their 7's. So now we add another rule for that special case.

But handcrafting rules is a tough job!

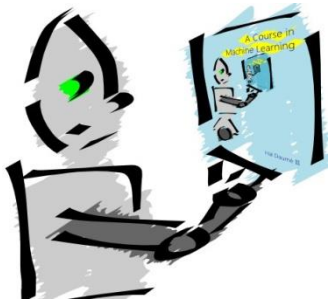
- This process of hand-crafting the rules to understand data is called feature engineering
- This term is also used to describe when we use the computer to find these features for us.
- It’s easy to overlook one rule, or even lots of them. It's a tough job !

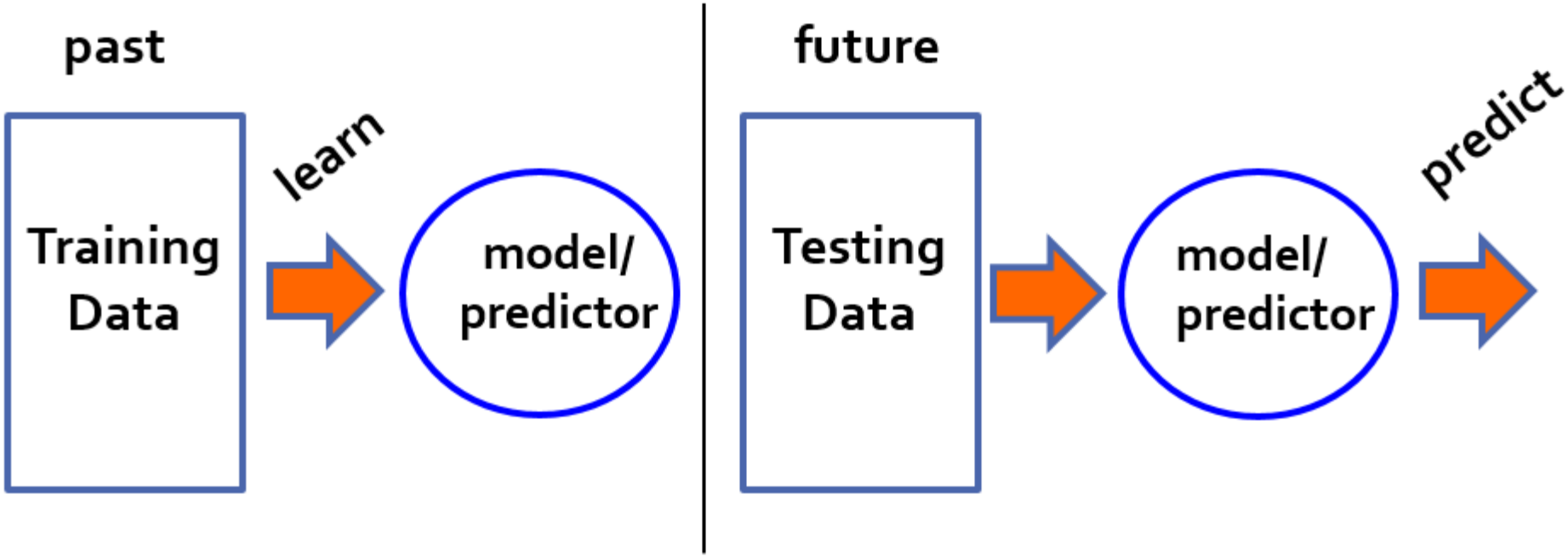
How does ML compare with Expert systems?

- **Expert Systems:** Difficult to manually find right set of rules, & make sure they work properly across a wide variety of data. These difficulties have doomed expert systems.
- **ML Systems:** Beauty is they learn a dataset's characteristics automatically.
- Don't have to tell an algorithm how to recognize a cat or dog, because system figures that out for itself.
- **Flip side of ML:** To do its job well, ML system often needs a lot of data. *Enormous amounts of data*.

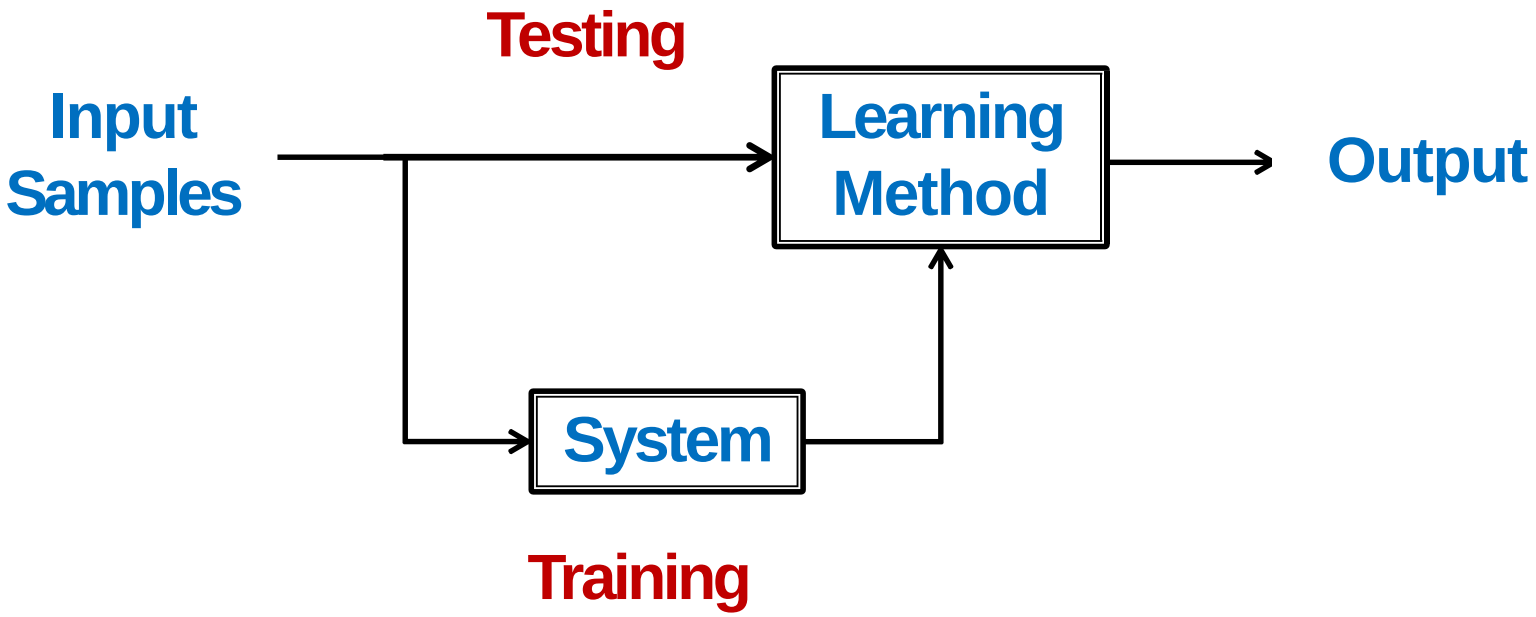
Machine Learning is…

Machine learning is about predicting the future based on the past.





Learning system model



Performance

- There are several factors affecting the performance:
  - Types of training provided
  - The form and extent of any initial background knowledge
  - The type of feedback provided
  - The learning algorithms used

What's a classifier?

- A classifier assigns a label to each sample describing which category or class, that sample belongs to.

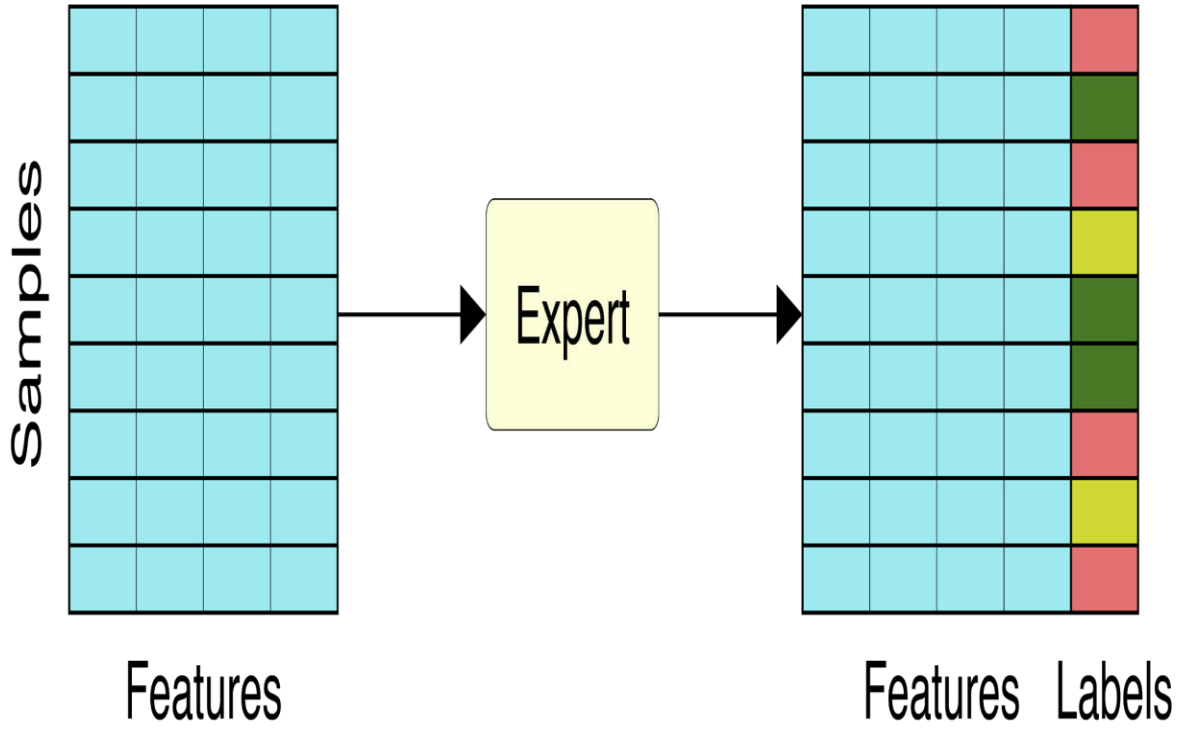
- Example of classifiers
  - If the input is a song, classifier assigns the label as the genre (e.g., rock or classical).
  - If it's a photo of an animal, the classifier assigns the label as the name of the animal shown (e.g., a tiger or an elephant).
  - In mountain weather for hiking, classifier may label the hiking experience into 3 categories: Lousy, Good, and Great.

Example of Samples, Features, Labels

- Weather measurements on a mountain for hiking
- Sample is weather at a given moment.
- Features are measurements: temperature, wind speed, humidity, etc.
- Hand over each sample (with a value for each feature) to a human expert.
- Expert examines features and provides a label for that sample.
- Expert's opinion, using a score from 0 to 100, tells how the day's weather would be for good hiking.
- Labels can be "Lousy", "Good", "Excellent" (weather for hiking)

Example of Samples, Features, Labels (Contd.)

- To label a dataset, we start with a list of samples, or data items.
- Each sample is made up of a list of features that describe it.
- We give the dataset to a human expert, who examines the features of each sample one by one, and assigns a label for that sample.

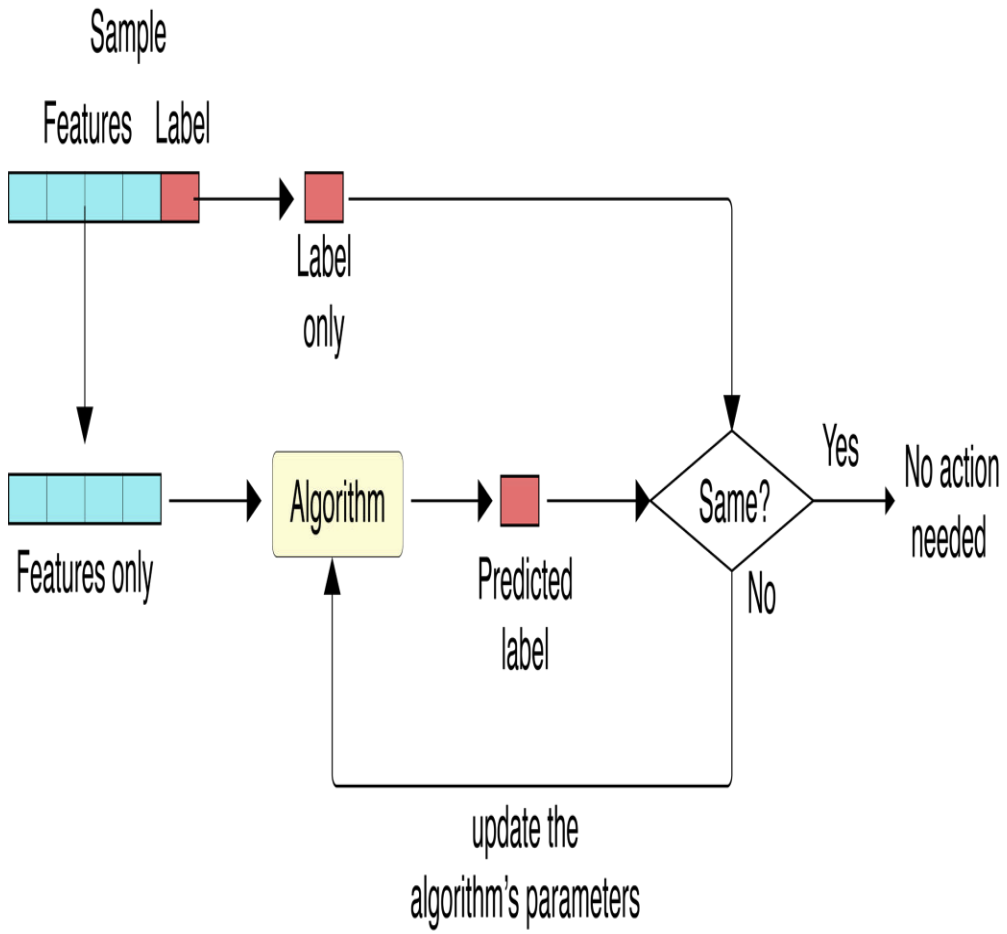


A Computerized Learning Strategy

- First, collect as much data as possible.
- Call each piece of observed data (say, the weather at a given moment) as sample.
- Call the names of the measurements that make it up (the temperature, wind speed, humidity, etc.) as features.
- Hand over each sample (with a value for each feature) to a human expert.
- Expert examines features and provides a label for that sample.
- Example: if our sample is a photo, the label might be the name of the person or the type of animal in the photo.

A Computerized Learning Strategy

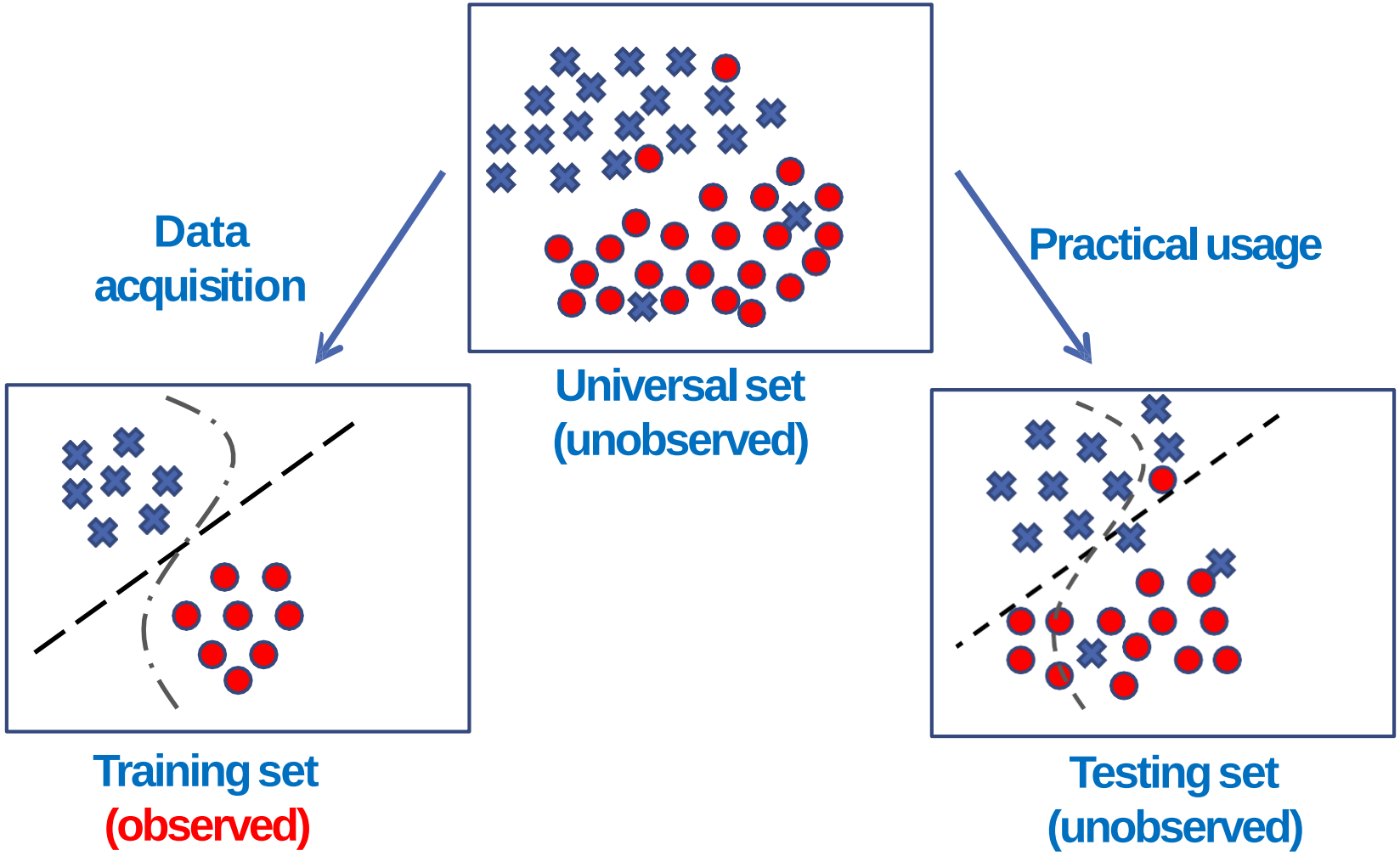
- Figure shows the idea of a learning strategy - one step of training or learning.
- Split the sample's features and its label. From the features, algorithm predicts a label.
- Compare prediction with truth label.
- If predicted label matches truth label, don't do anything.
- Otherwise, tell algorithm to update itself
- The process is basically trial and error.



Data for Training & Testing

- First, set aside some of these labeled samples for time being (Test).
- Give remaining labeled data to our computer, and ask it to find a way to come up with the right label for each input.
- We do not tell it how to do this.
- Instead, we give labelled data to an algorithm with a large number of parameters it can adjust (perhaps even millions of them).
- Different types of learning will use different algorithms.

Training and Testing

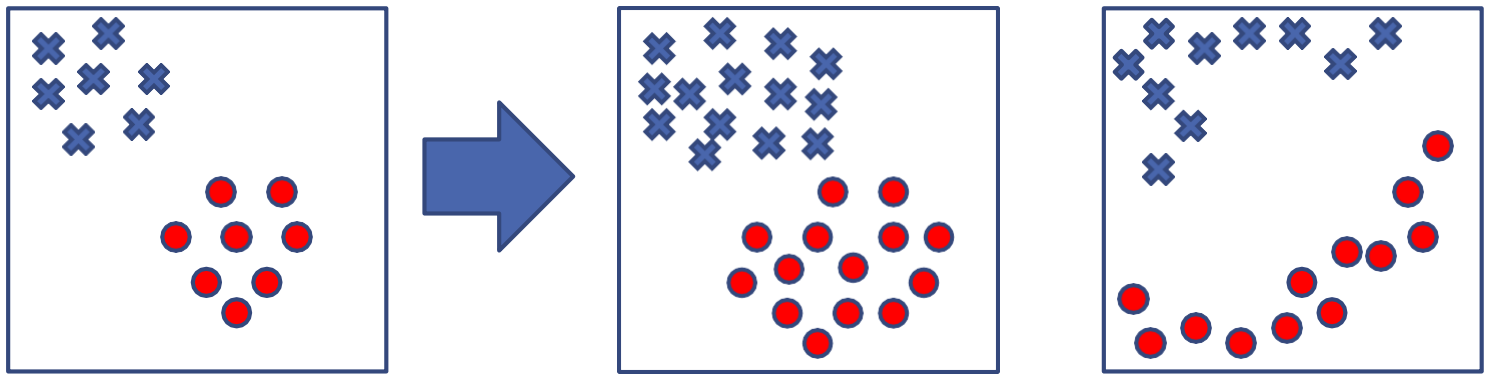


Training data

- Training is the process of taking a system that's been initialized with default or random values, and reconfiguring it so that it's tuned to the data we want to understand.
- Training data is data used to train the system.
- Real-world data is data the classifier will see in the real world, once it's been released, or deployed.
- Real-world data will only arrive after system has been deployed.

Training and testing

- **Training** is the process of making the system able to learn.
  - Training set and testing set come from the same distribution
  - Need to make some assumptions or bias



Parameters and hyper parameters

- Learning algorithm modifies itself its own parameter values, over time.
- Learning algorithm are also controlled by values that we set (such as the learning rate we saw above).
- These are called hyper parameters.

Difference between parameters and hyper parameters

- Computer adjusts its own parameter values during the learning process, while we specify the hyper parameters when we write and run our program.

Training step and Learning rate

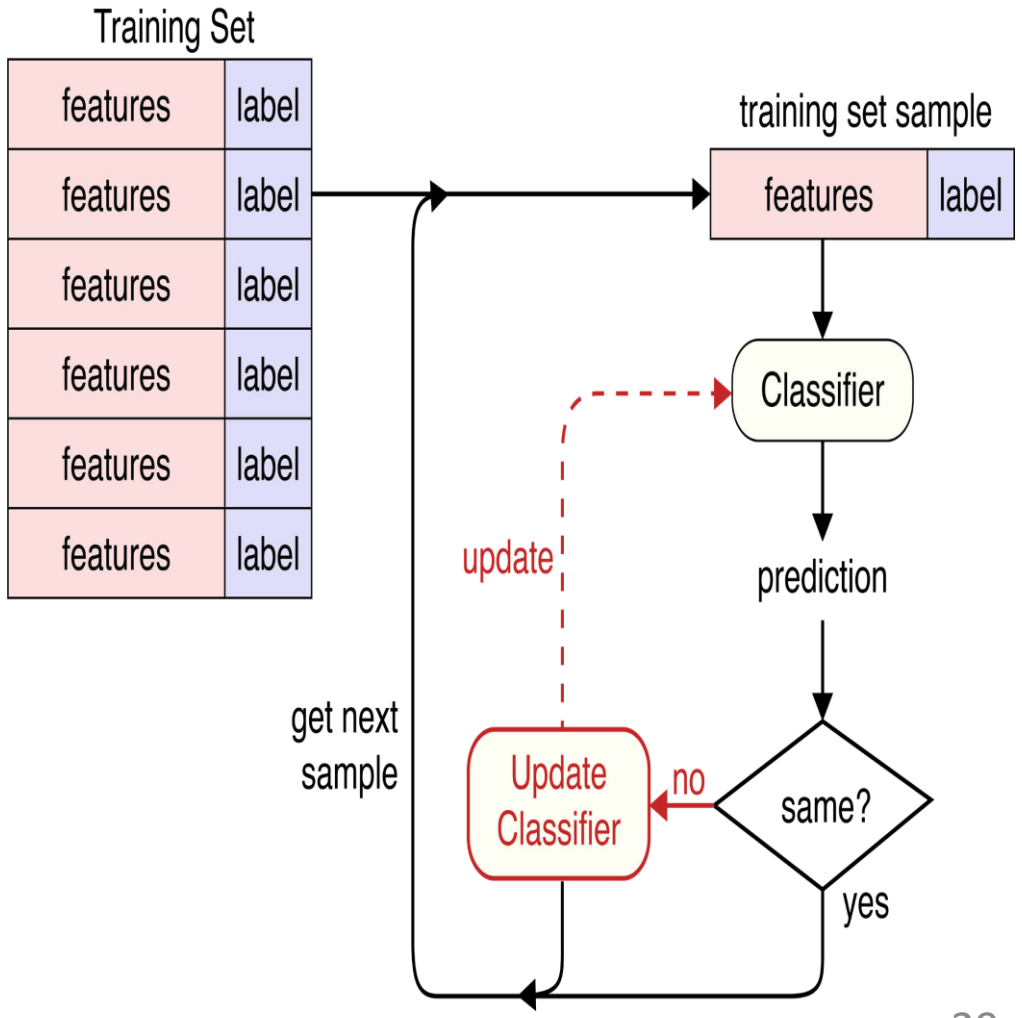
- Each algorithm learns by changing the internal parameters it uses to create its predictions.
- Big change: risk of changing them so much that it makes other predictions worse.
- Small change: Cause learning to run slower.
- We have to find by trial and error for each type of algorithm the right trade-off between these extremes.
- We call the amount of updating the learning rate,
- A small learning rate is cautious and slow,
- A large learning rate speeds things up but could backfire.

Testing

- We now return to the labeled data kept aside in the last section.
- This is called as test data.
- We evaluate how the system can generalize what it learned, by showing these samples that it's neverseen before.
- This test set shows how the system performs on new data.

Lets train and test a Categorizer (for example)

- Present each sample in training set to classifier, one at a time.
- For each sample, give the sample's features and ask it to predict its category.
- If prediction is correct, then move to next sample.
- If prediction is wrong, feed classifier's output and correct label back to classifier.
- ML algorithm uses correct label, predicted label, and the current state of classifier to modify the classifier's internal parameters - so that it's likely to predict correct label if it sees this sample again.

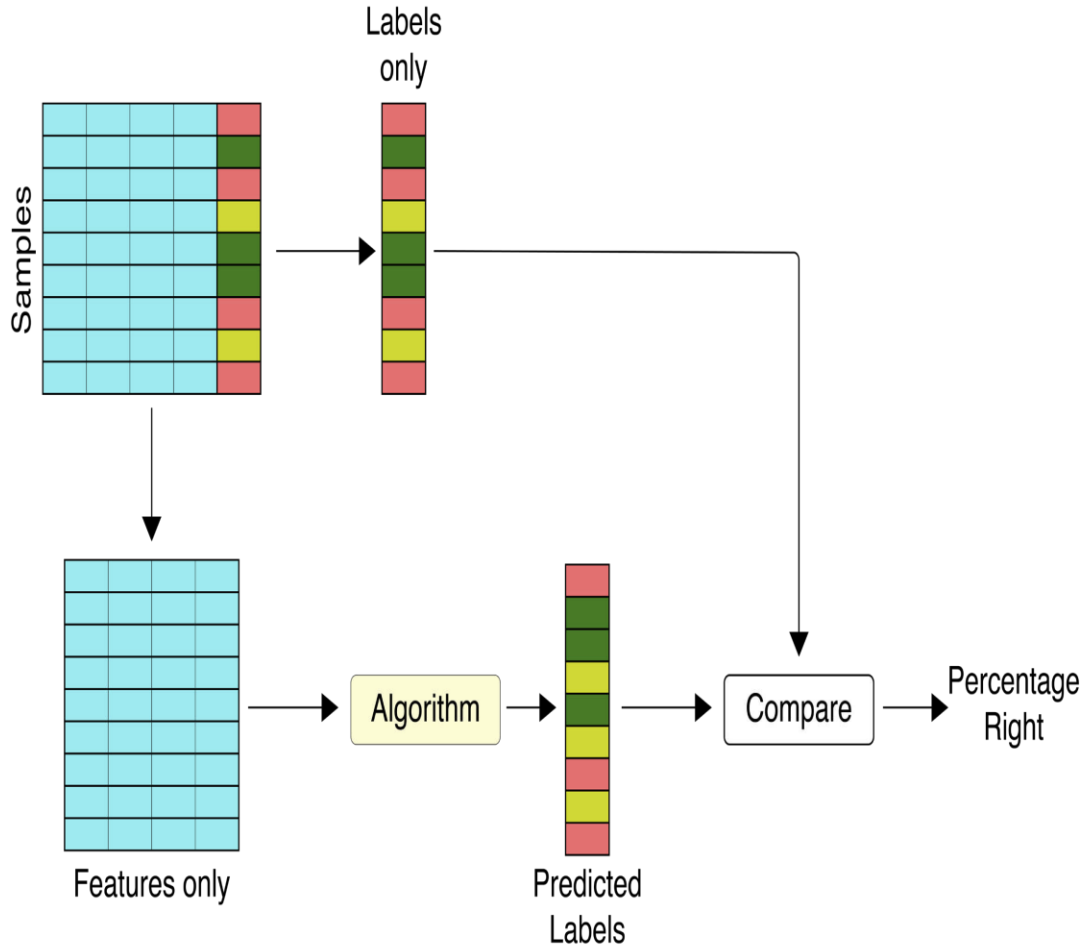


How long to train? Epochs

- Epoch is a run through the entire training set.
- Usually you must train through many epochs.
- Keep training as long as the system is still learning and improving its performance on the test data.
- Stop when system's performance is good enough for the task.
- Stop also if you are out of time.

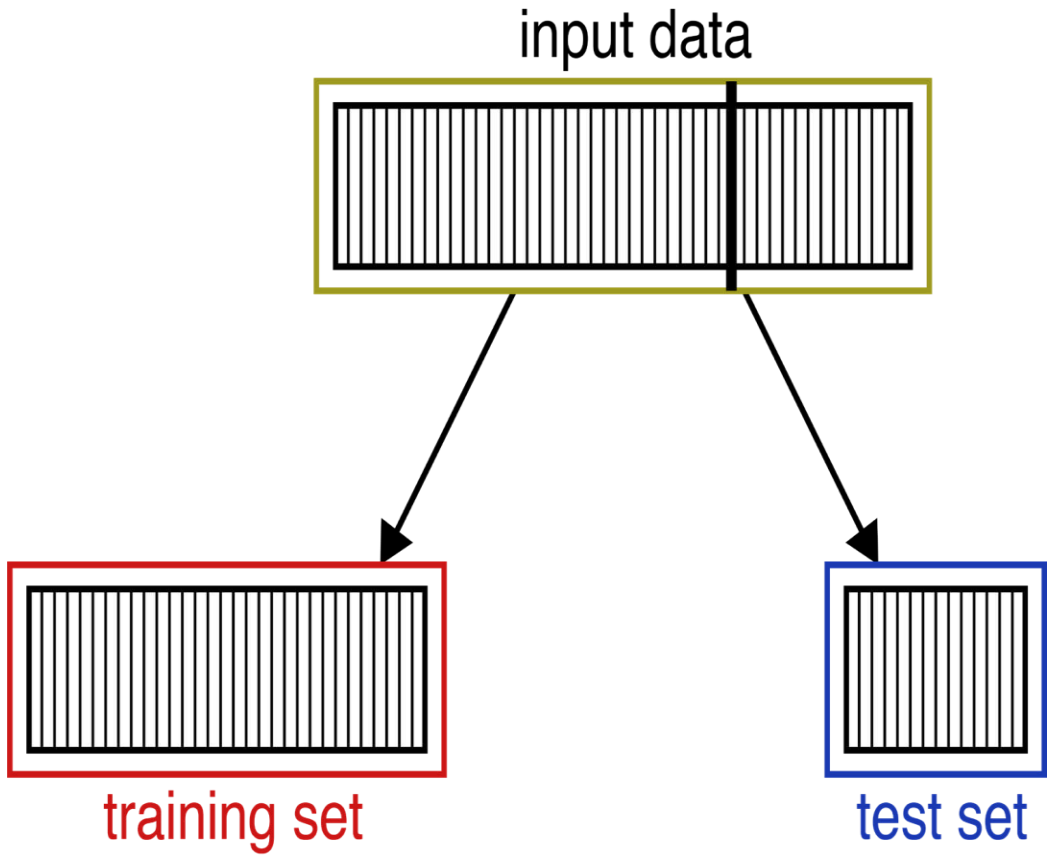
Testing – Procedure for evaluating a classifier

- Split the test data (not training data) into features and labels.
- Algorithm predicts a label for each set of features.
- Compare predictions with the
- truth labels to get a measurement of accuracy.
- If it's good enough, deploy the system.
- If the results aren't good enough, go back and train some more.
- In this evaluation process there is n o feedback and no learning.



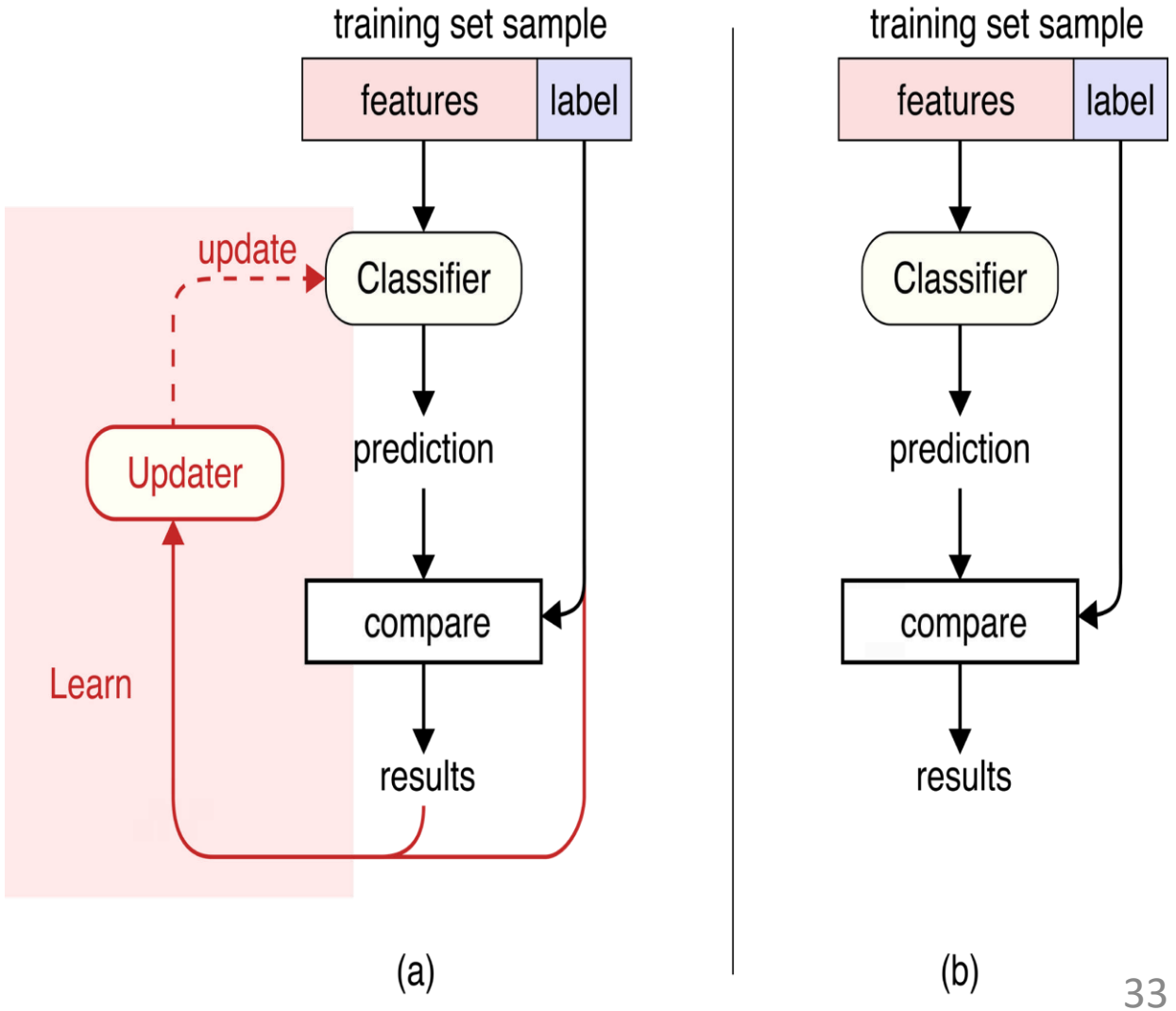
How to create training and test data sets?

- Create test data by splitting original data collection into two pieces: training set and test set.
- Split to give about 75% of the samples to training set.



Difference between training and testing

- In training, compare the prediction and the real label, and use any error between them to make classifier learn to do better.
- In testing, omit learning step.



How to Retrain, if testing results are not good

- Use again original training set data. Note that these are the same samples.
- Shuffle this data first - but no new information.
- Show every sample again, letting it learn along the way again.
- Computer learns over and over again from the very same data.
- Now, show test data set .
- Ask algorithm to predict labels for the test set again.
- If the performance isn't good enough, go back to original training set again, and then test again.
- Repeat this process often hundreds of times. Let it learn just a little more each time.
- Computer doesn't get bored or cranky seeing the same data over and over.

When do we deploy the System ?

- When the algorithm has learned enough to perform well enough on the test set that we're satisfied, we're ready to deploy, or release, our algorithm to the world.
- Users submit data and our system returns the label it predicts.
- That's how pictures of faces are turned into names, sounds are turned into words, and weather measurements are turned into forecasts.

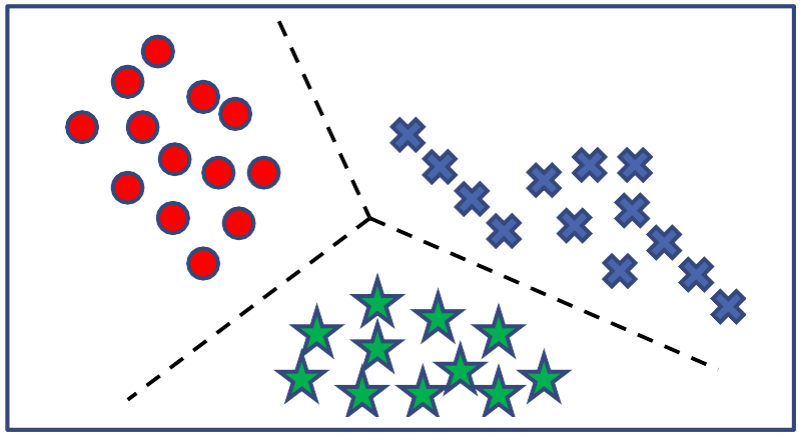
Machine Learning – Major categories-Algorithms

- The success of machine learning system also depends on the algorithms.
- The algorithms control the search to find and build the knowledge structures.
- The learning algorithms should extract useful information from training examples.

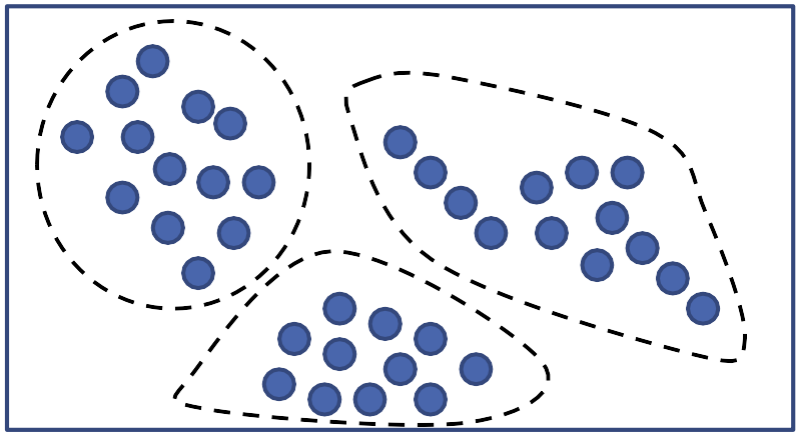
ML Algorithms

- Supervised learning
  - Prediction
  - Classification (discrete labels), Regression (real values)
- Unsupervised learning
  - Clustering
  - Probability distribution estimation
  - Finding association (in features)
  - Dimension reduction
- Semi-supervised learning
- Reinforcement learning
  - Decision making (robot, chess machine)

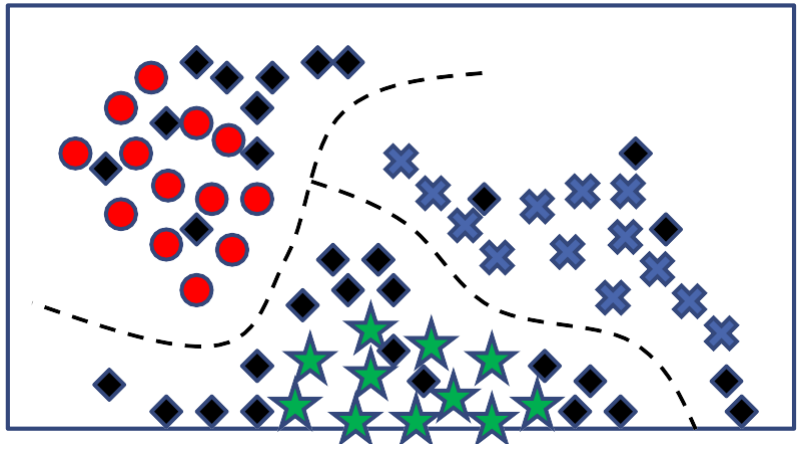
Algorithms



Supervised learning



Unsupervised learning



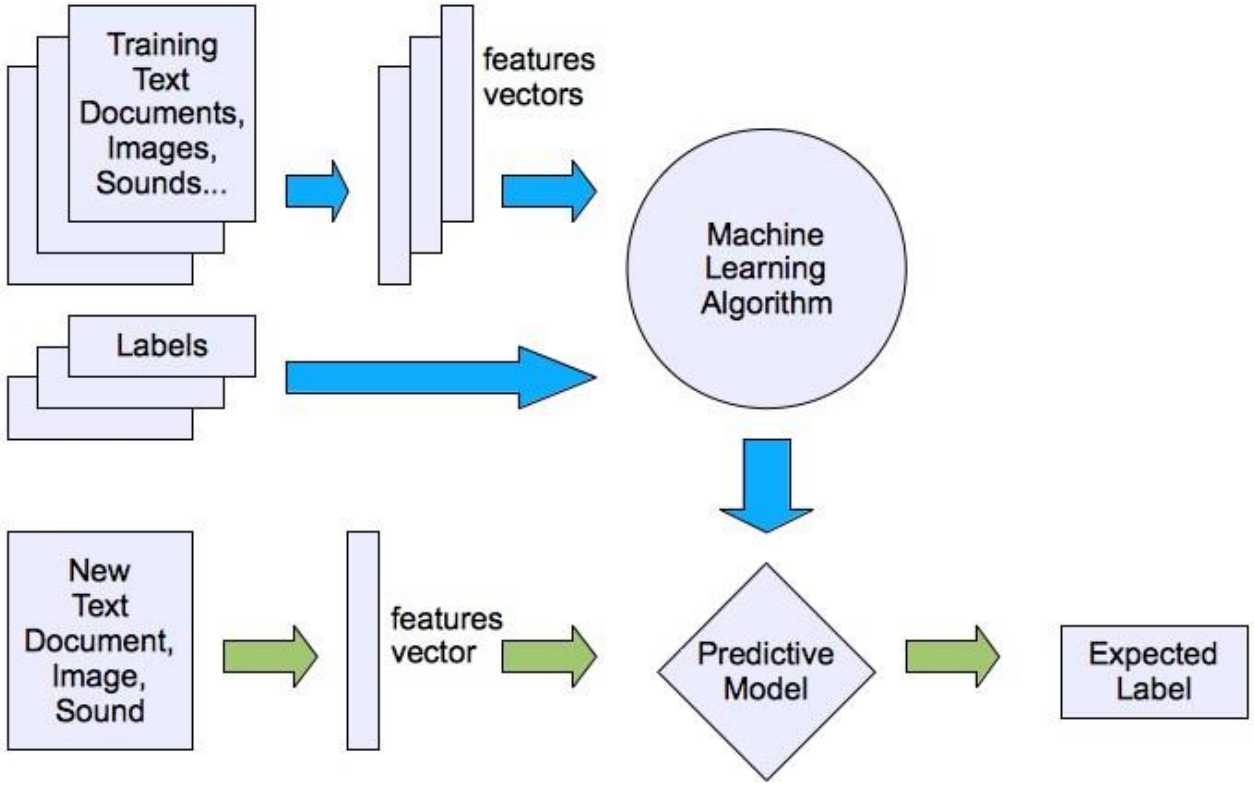
Semi-supervised learning

Supervised Learning (ML)

- Supervised learning (SL) is done for samples with pre-assigned labels.
- Supervision comes from the labels.
- Labels guide the comparison step.
- There are two general types of supervised learning, called classification and regression.
- Classification: look through a given collection of categories to find the one that best describes a particular input.
- Regression: take a set of measurements and predict some other value

Machine learning structure

- **Supervised learning (SL)**



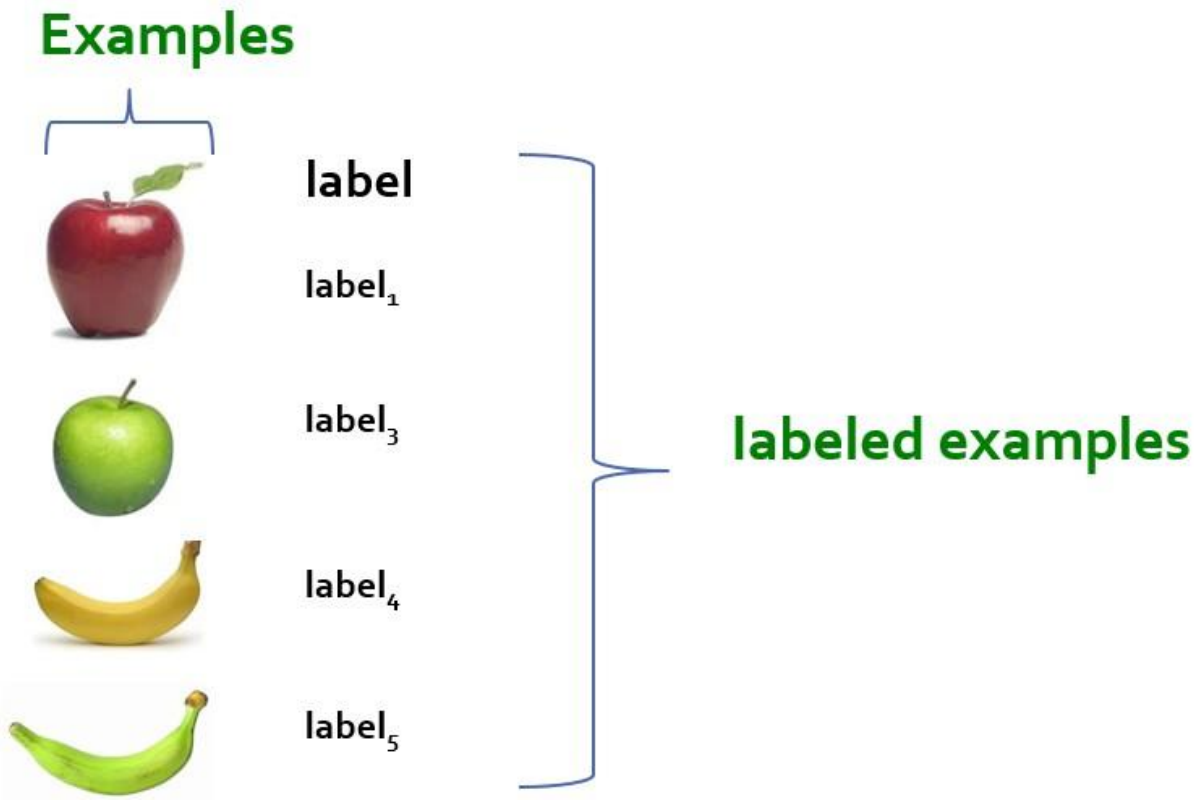
SL – Classification

- Start training by providing a list of all the labels (or classes, or categories) that we want it to learn.
- Make the list so that it has all the labels for all the samples in the training set, with the duplicates removed.
- Train the system with lots of photos and their labels, until it does a good job of predicting the correct label for each photo.
- Now, turn the system loose on new photos it hasn't seen before.
- For those objects it saw during training, It should properly label images.
- Caution: For those objects it did not see during training, the system will try to pick the best category from those it knows about.
- Next Figure shows the idea.

SL – Classification Example

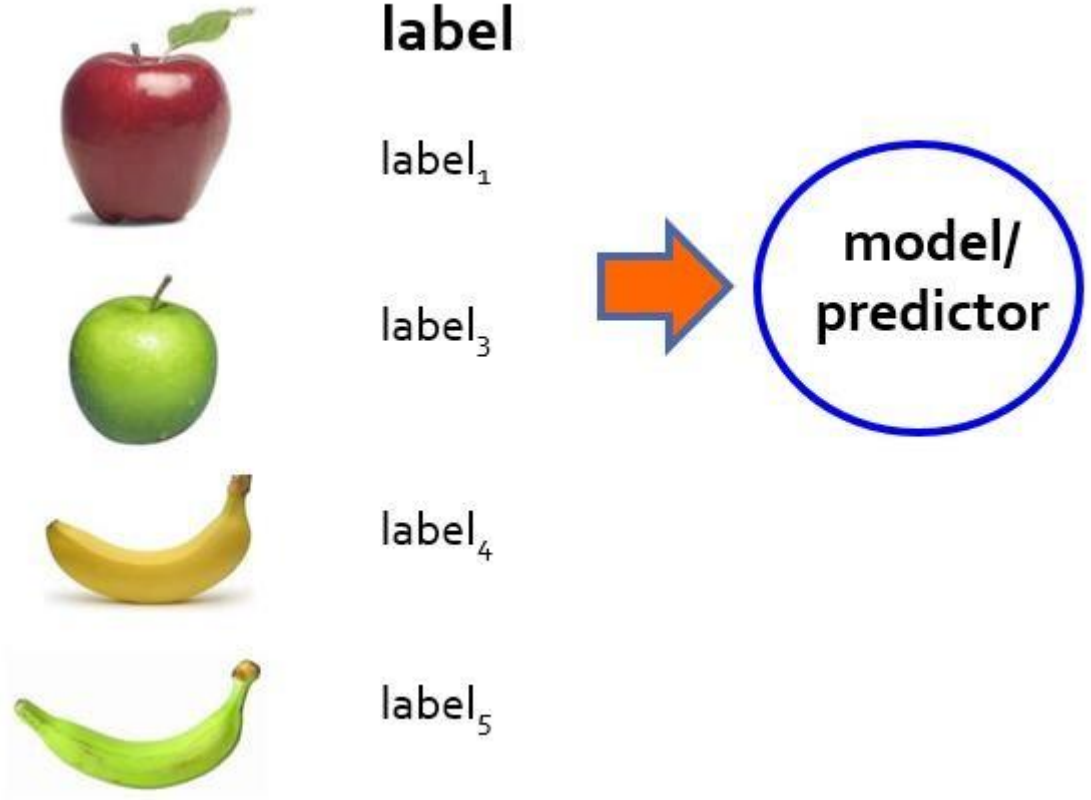
- **Example: Sort and label photos of everyday objects.**
- **We want to sort them : an apple peeler, a salamander, a piano, and so on.**
- **We want to classify or categorize these photos.**
- **The process is called classification or categorization.**

SL – Classification Example



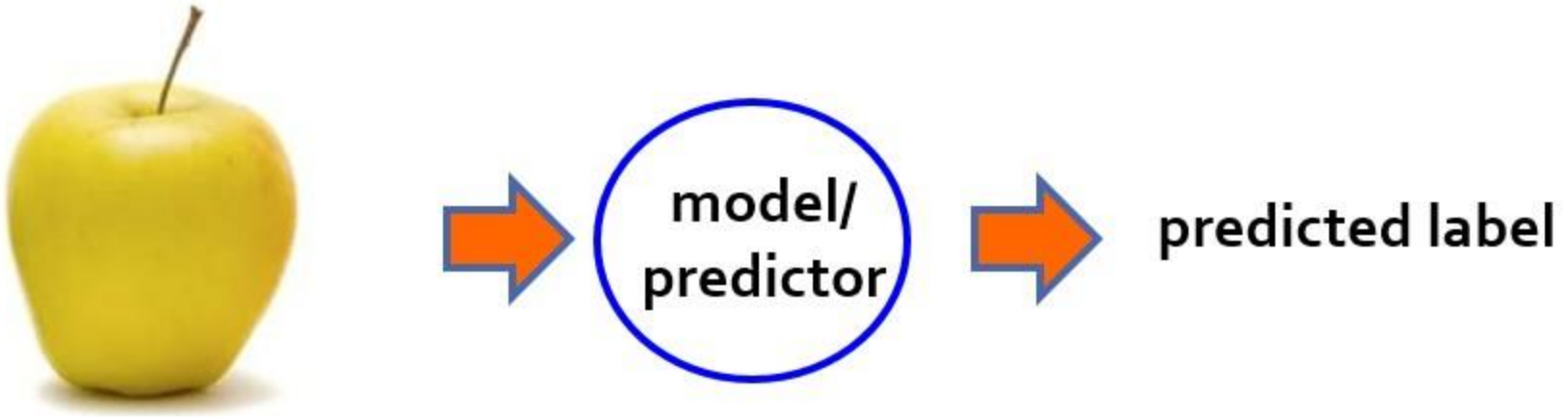
Supervised learning: given labeled examples

SL – Classification Example



Supervised learning: given labeled examples

SL – Classification Example



Supervised learning: learn to predict new example

SL – Classification Example

label

apple

apple

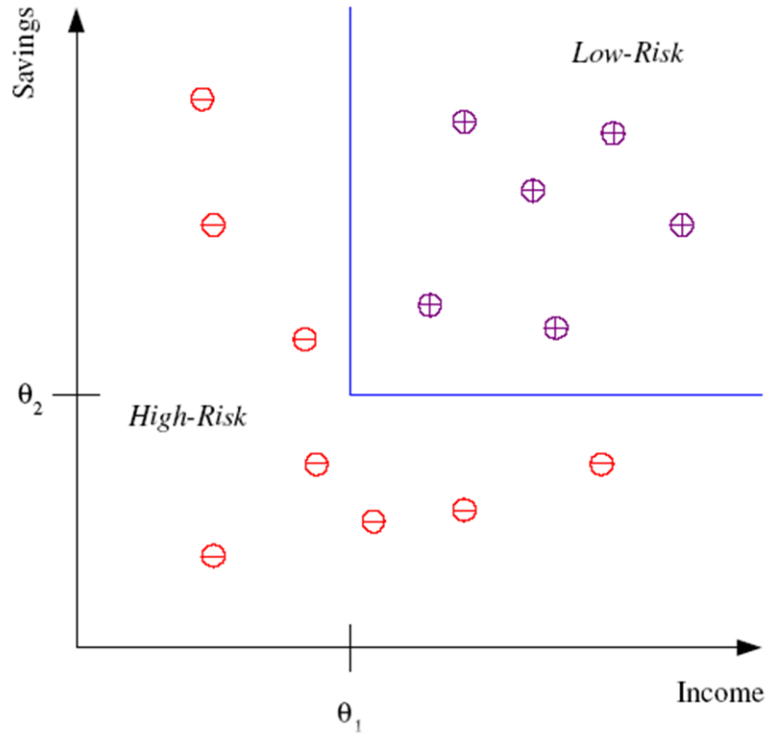
banana

banana

Classification: a finite set of labels

SL – Classification Example

Differentiate between low-risk and high-risk customers from their *income* and *savings*



SL – Classification Example

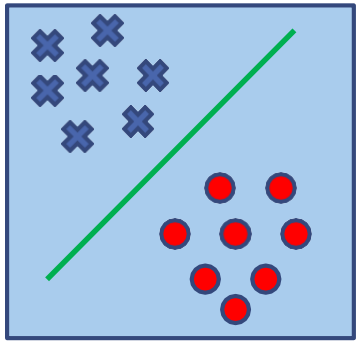
- In Figure, we used a trained classifier to identify four images never seen before.
- The system had not been trained on metal spoons or headphones,
- In both cases it found best match it could.
- To correctly identify those objects, the system needs to see multiple examples of them during training.

Learning techniques

- **Supervised learning categories and techniques**
  - **Linear classifier (numerical functions)**
  - **Parametric (Probabilistic functions)**
    - **Naïve Bayes, Gaussian discriminant analysis (GDA), Hidden Markov models (HMM), Probabilistic graphical models**
  - **Non-parametric (Instance-based functions)**
    - ***K*-nearest neighbors, Kernel regression, Kernel density estimation, Local regression**
  - **Non-metric (Symbolic functions)**
    - **Classification and regression tree (CART), decision tree**
  - **Aggregation**
    - **Bagging (bootstrap + aggregation), Adaboost, Random forest**

Learning techniques

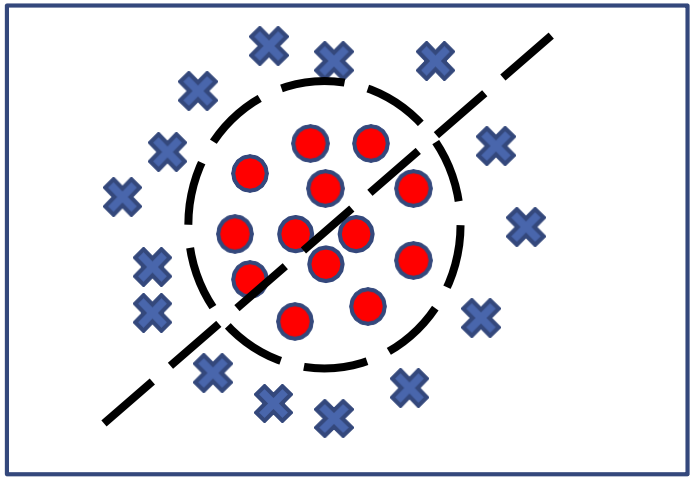
- Linear classifier



- Techniques:
  - Perceptron
  - Logistic regression
  - Support vector machine (SVM)
  - Ada-line
  - Multi-layer perceptron (MLP)

Learning techniques

- **Non-linear case**



- **Support vector machine (SVM):**
  - **Linear to nonlinear:** Feature transform and kernel function

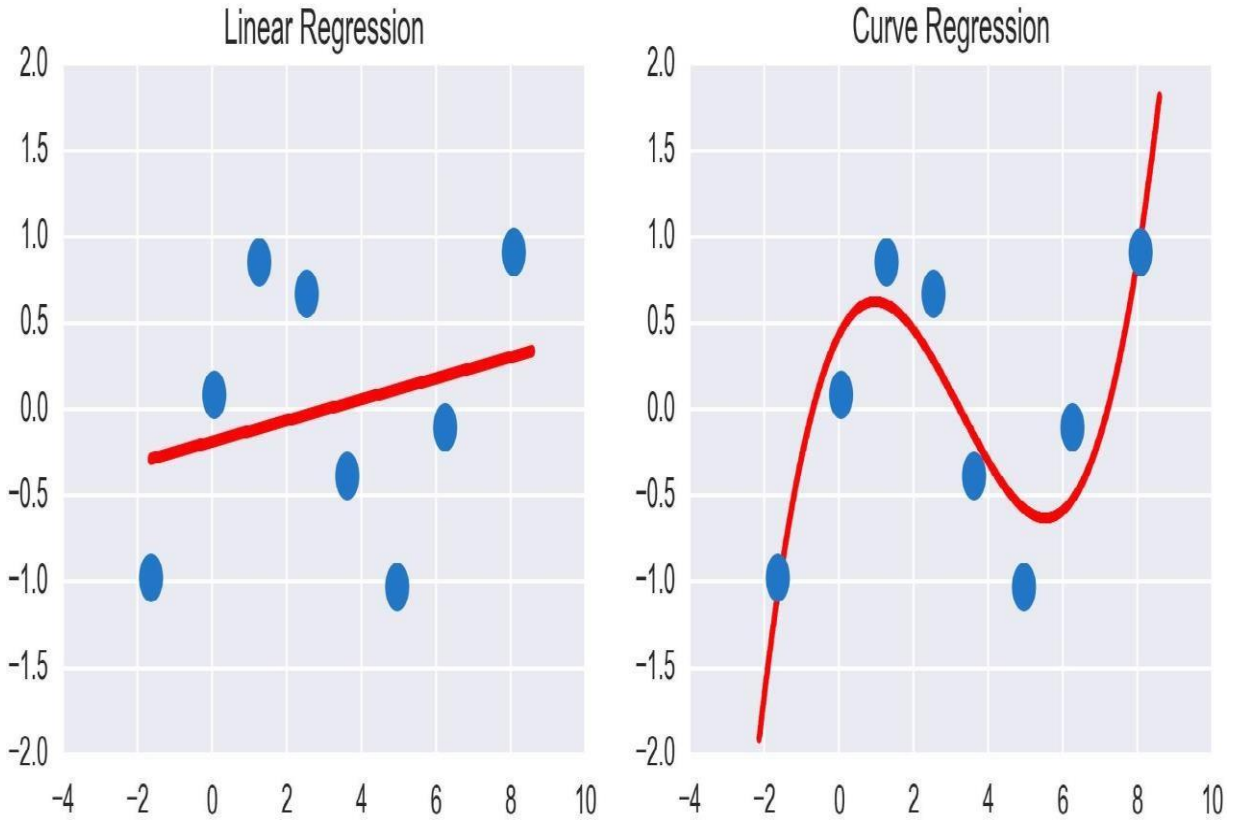
Classification Applications

- **Face recognition**: Pose, lighting, occlusion (glasses, beard), make-up, hair style

- **Character recognition**: Different handwriting styles.
- **Speech recognition**: Temporal dependency.
- **Medical diagnosis**: From symptoms to illnesses
- **Web Advertizing**: Predict if a user clicks on an ad on the Internet.
- **Biometrics**: Recognition/authentication using physical and/or behavioral characteristics: Face, iris, signature, etc

SL - Regression

- Regression is process of filling in or predicting data.
- "Regression" uses statistical properties of the data to estimate missing or future values.
- Most famous kind of regression is linear regression.
- Left: Linear regression fits a straight line (red) to the data (blue). The line is not a very good match to the data, but it has the benefitof being simple.
- Right: Nonlinear regression fits a curve to same data. This is a better match to the data, but has more complicated form and requires more work (and thus more time)



Supervised learning: Regression

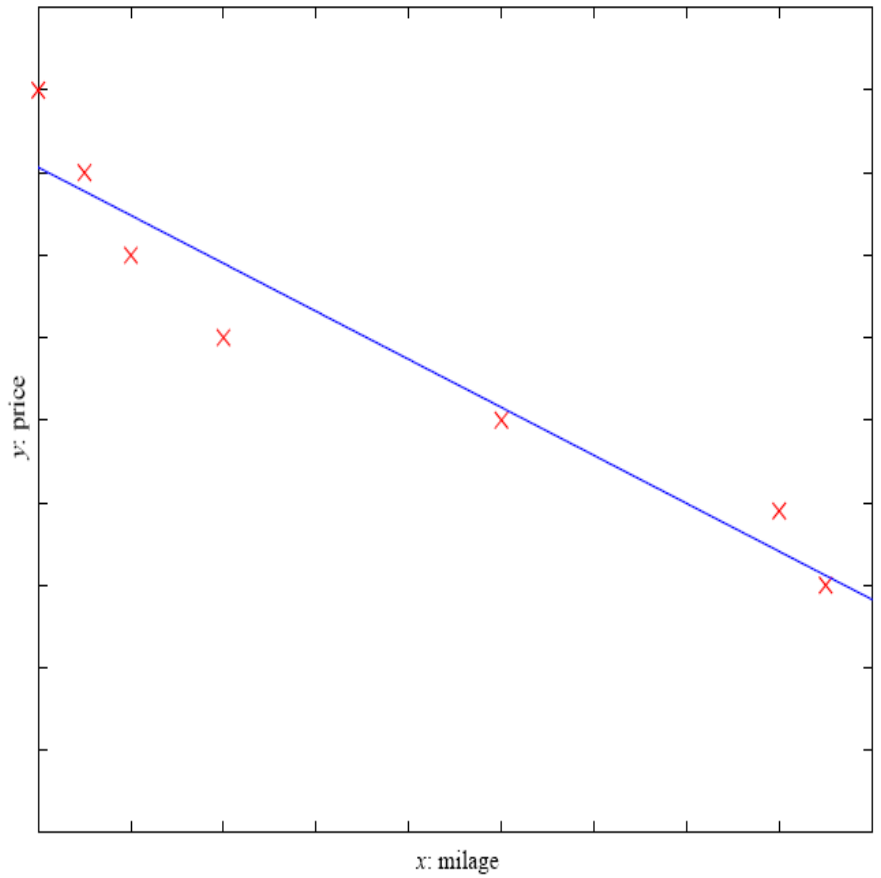
	label
	-4.5
	10.1
	3.2
	4.3

Regression: label is real-valued

Regression Example

- Price of a used car

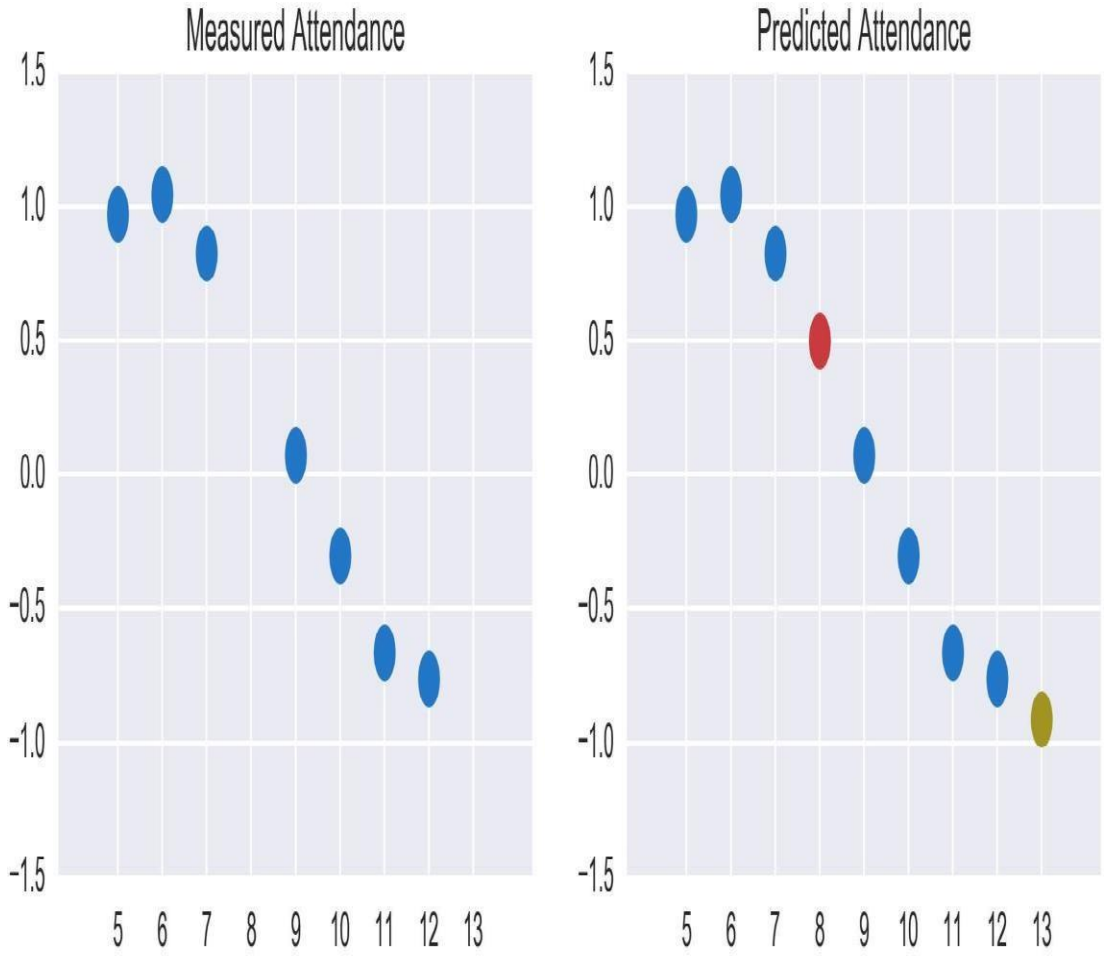
x : car attributes (e.g. mileage)
y : price



Regression Example

Example: Music band attendance

- Problem: An incomplete collection of measurements of attendance.
- Estimate missing values of attendance.
- Data: Attendance at a series of concerts at a local arena.
- Problem: Unfortunately, we lost count for one evening's performance.
- Also want to know what tomorrow's attendance is likely to be.



Regression Applications

- Economics/Finance: predict the value of a stock
- Epidemiology
- Car/plane navigation: angle of the steering wheel, acceleration
- Temporal trends: weather over time

Supervised learning: Ranking

label

1

4

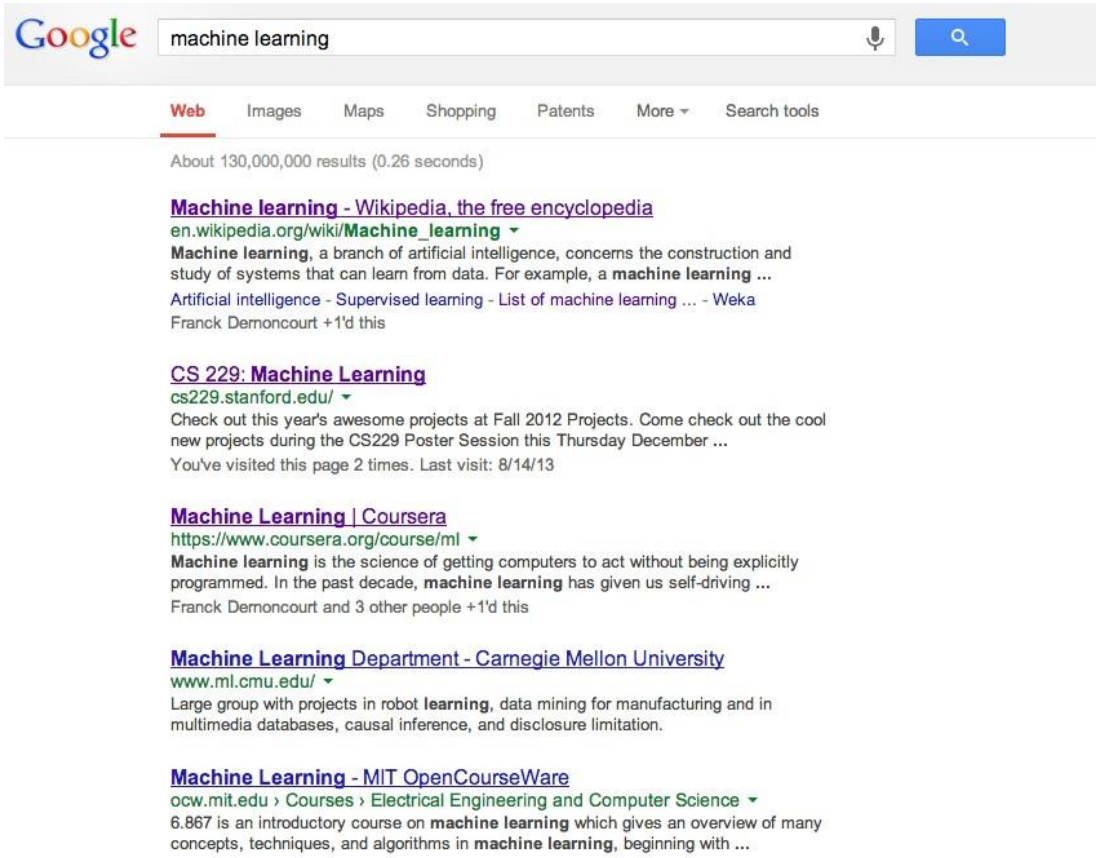
2

3

Ranking: label is a ranking

Ranking example

- Given a query and a set of web pages, rank them according to relevance



Ranking Applications

- User preference, e.g. Netflix “My List” -- movie queue ranking
- iTunes
- flight search (search in general)
- reranking N-best output lists

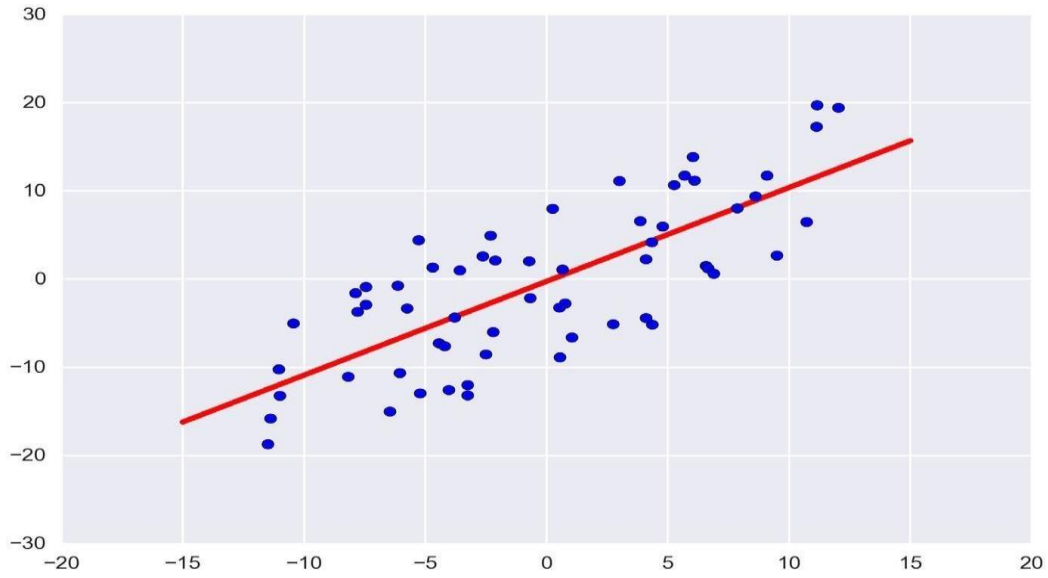
DEEP LEARNING

Deep learning

- More recently, the phrase deep learning was coined to refer to
  - Approaches to machine learning that use specialized layers of computation, stacked up one after the next.
- This makes a "deep" structure, like a stack of pancakes.
- Deep learning (DL) refers to the nature of the system we create, not to any particular algorithm.
- DL really refers to this particular style or approach to machine learning (ML).

Compare ML & DL - Fit the best line

- **ML:** Find the best straight line through a bunch of data points, see Figure.
- **DL:** Given a set of data points (in blue), we can imagine a straightforward algorithm that computes the best straight line (in red) through those points.



ML vs DL – Line fitting example

- ML: Represent a straight line with just a few parameters.
- Use some formula to compute parameter values, given the input data points.
- This is a familiar algorithm, uses analysis to find the best way to solve a problem.
- Strategy used by many ML algorithms.

- DL: Don't know how to directly calculate the right answer, so we build a system that can figure out how to do that itself.
- For DL, we create an algorithm that can work out its own answers, rather than implementing a known process that directly yields an answer
- DL learns slowly. Every time program sees a new piece of data, it improves its own parameters
- DL ultimately finds a set of good values.
- Much more open-ended than the one that fits a straight line.

Heart of DL success

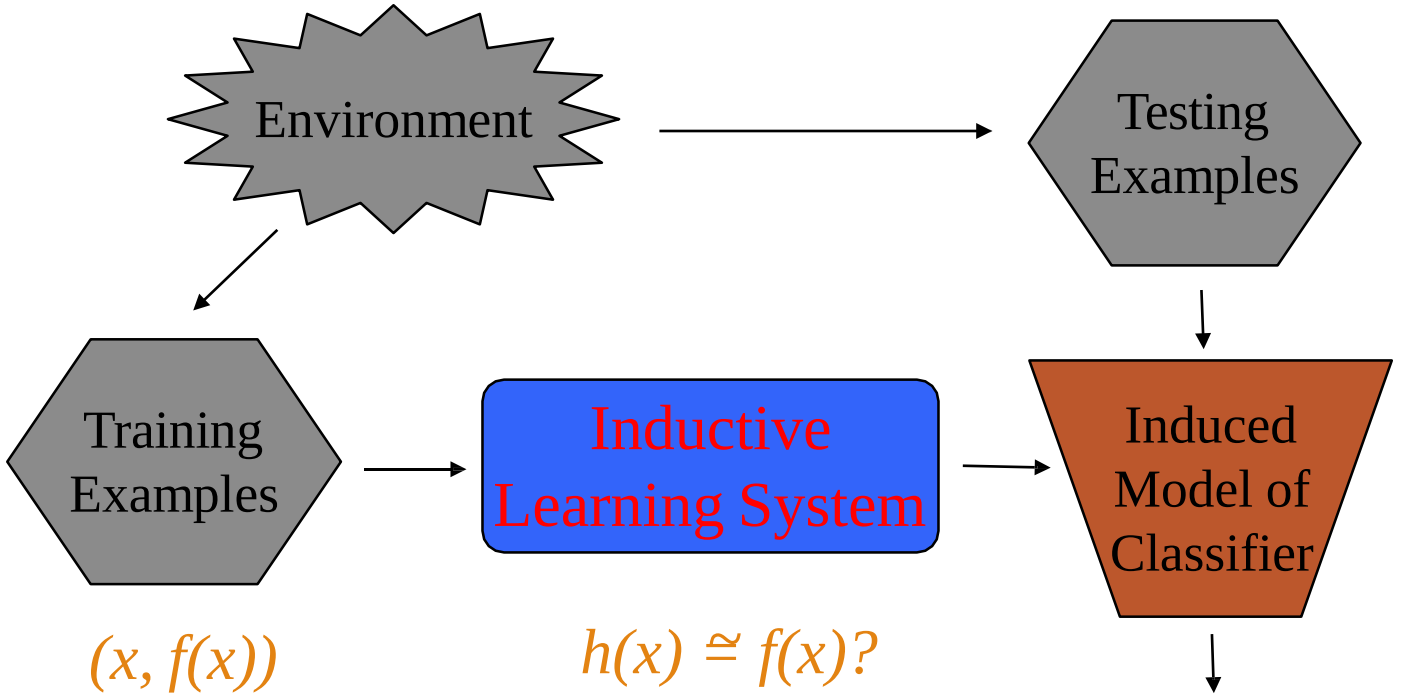
- What's a main reason for the enormous success of deep learning algorithms?

- DL success is due to the programs that find their own answers (apart from availability of high performance computing power).

Neural Networks

Classification Systems

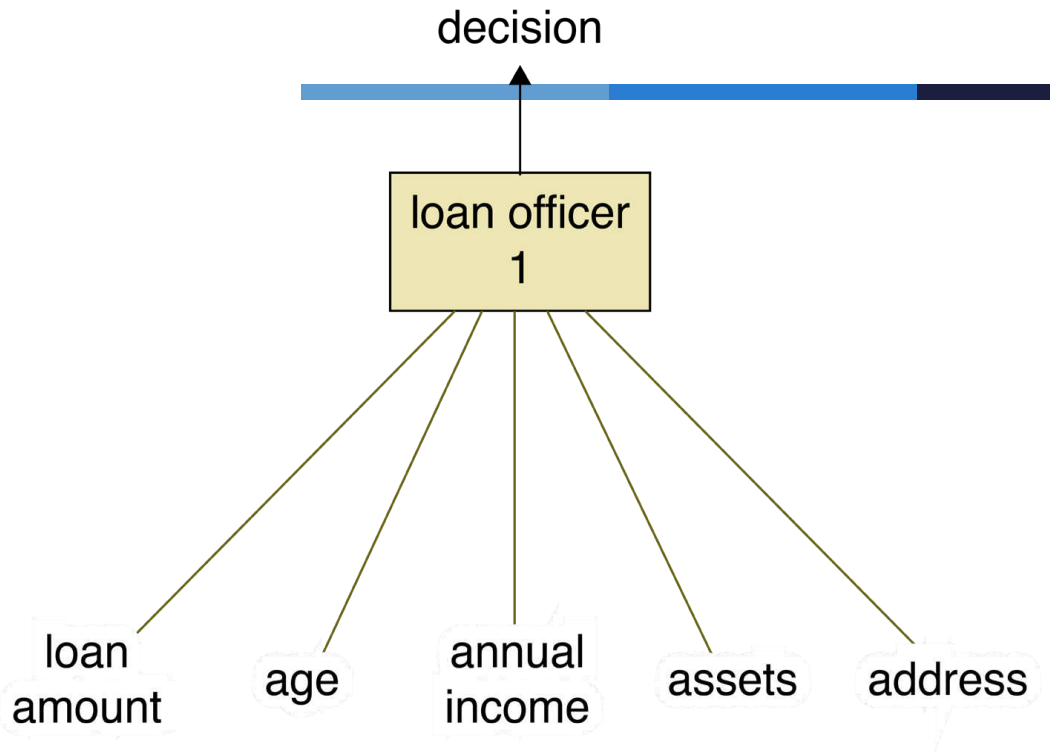
Basic Framework for Learning



A problem of representation and search for the best hypothesis, h(x).

Bank Loan Example with Single layer Neural Network

- Let's consider the process of getting a loan.
- In practice, any loan application is going to be a complex affair.
- Based on the applications for those loans, they came up with rules that would let them predict whether a loan was likely to end up getting paid back or not.
- This is of course just like what a perceptron does. The inputs are weighted and combined to produce a final score, as in Figure.
- Say, loan is rejected

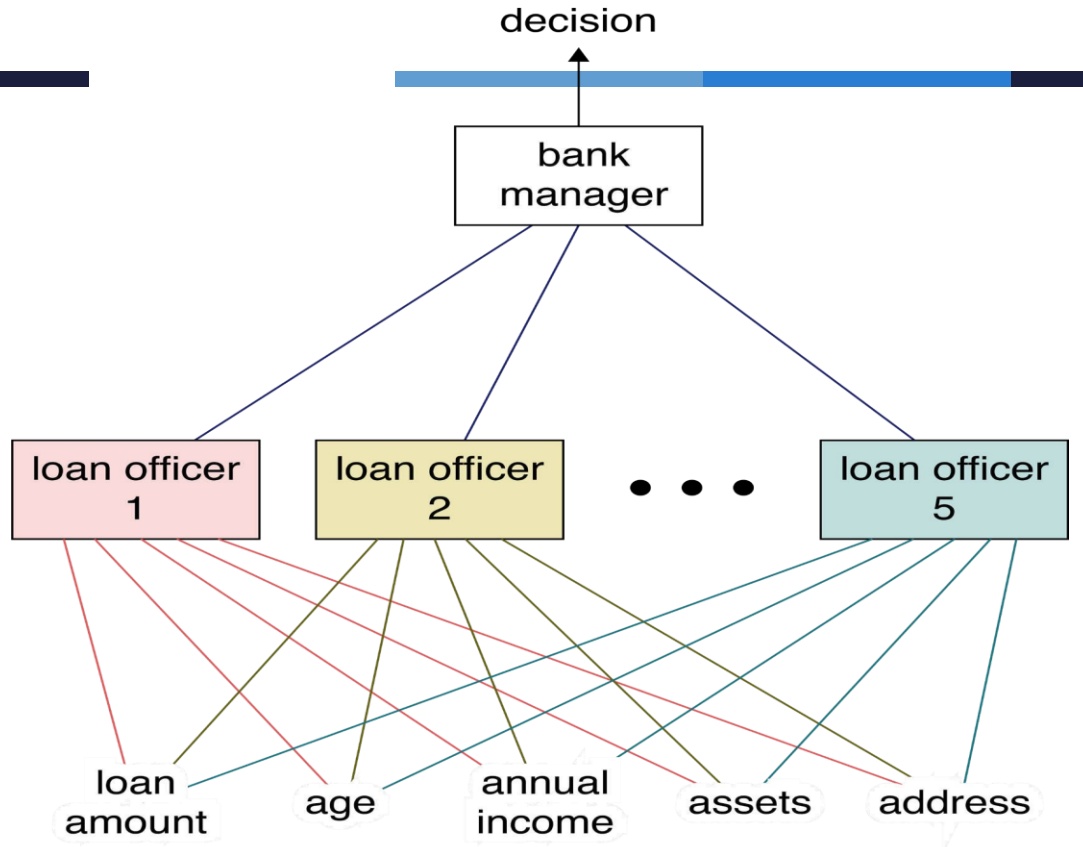


Bank Loan Example with 2 layer Neural Network

We might try our luck at another bank.

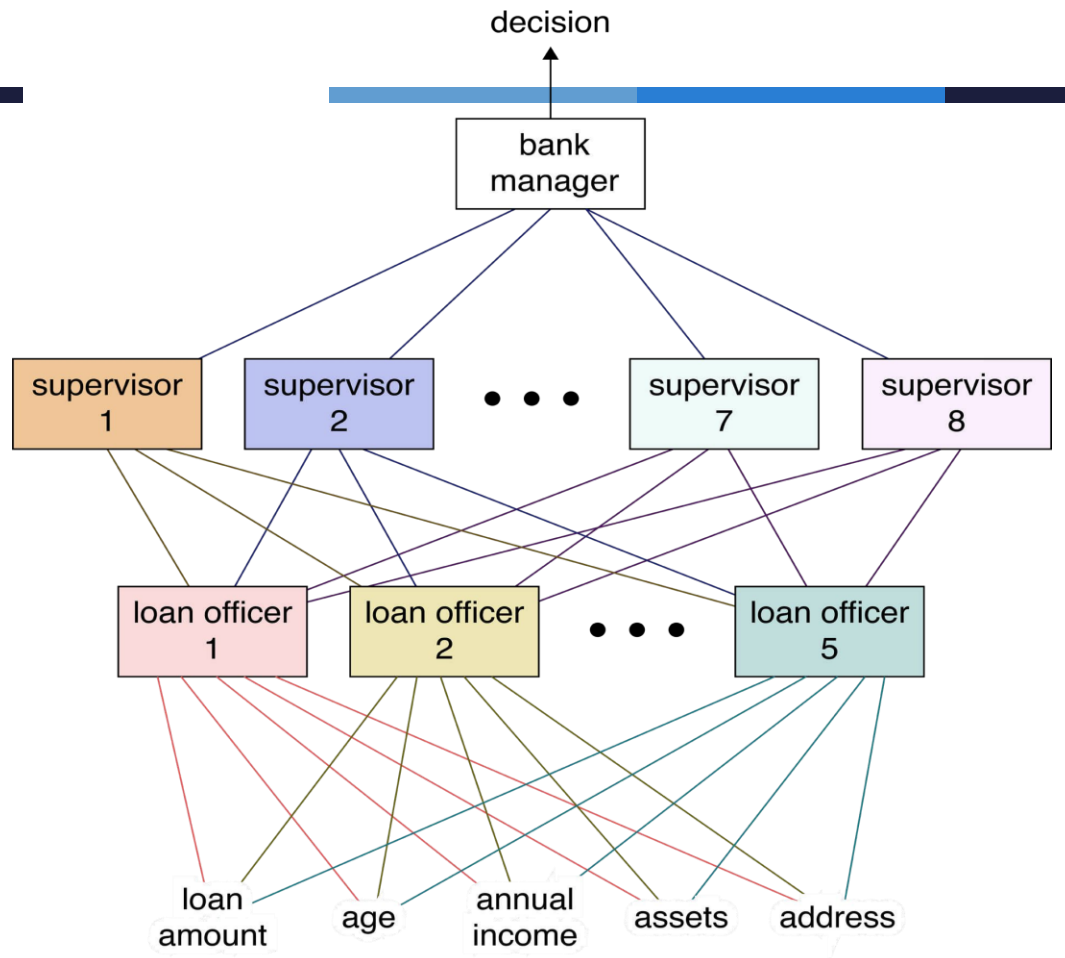
In this bank, suppose that there are 5 different loan officers, and each one has developed his own idiosyncratic procedure for evaluating the criteria that go into a loan.

We can draw this as a two-layer neural network.

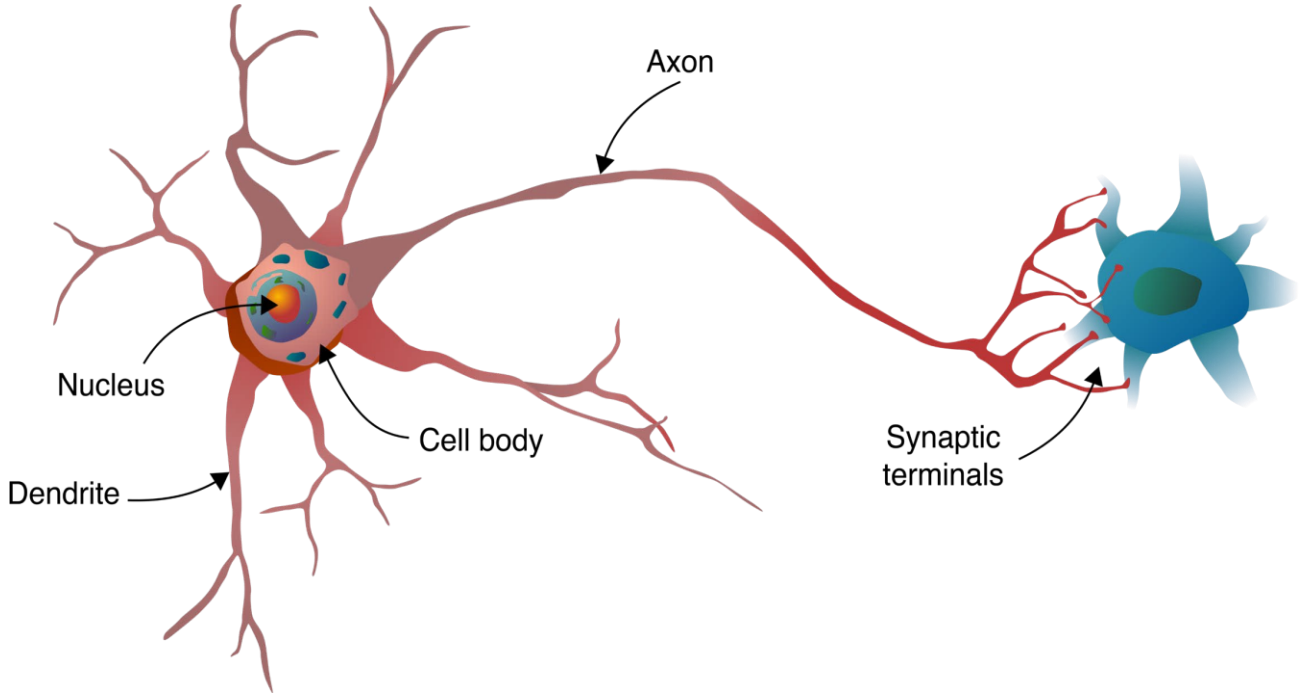


Bank Loan Example with 3 layer Neural Network

- Suppose bank hires a bunch of ssupervisors to sit between the loan officers and bank manager.
- Suppose we have 5 loan officers, 8 supervisors, and 1 bank manager.
- Then we have a 3-layer neural network to represent this process, as in Figure.
- Supervisors look at the decisions of the loan officers.
- Supervisors combine the results of the loan officers, and then pass their judgements up to the bank manager.
- Bank manager final decision is based on how he chooses to weight the conclusions of the supervisors.



Neurons



- Neurons are the nerve cells that make up the brain, used for our cognitive abilities.
- Neurons are information processing machines.

A sketch of a biological neuron (in red) with a few major structures identified. This neuron's outputs are communicated to another neuron (in blue), only partially shown.

How real biological neurons inspired the artificial neurons we use in machine learning, and how those little bits of processing work alone and in groups.

Neuron Model

Neuron collects signals from *dendrites*

Sends out spikes of electrical activity through an *axon*, which splits into thousands of branches.

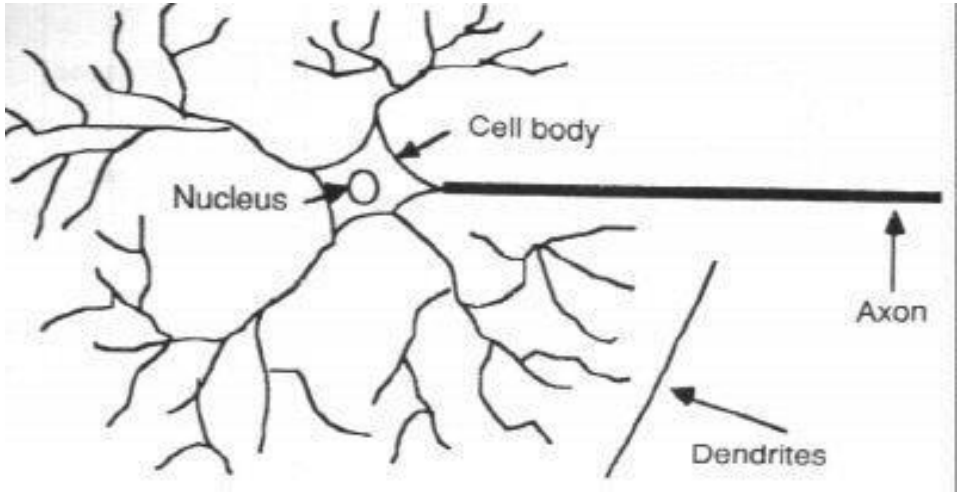
At end of each brand, a *synapses* converts activity into either exciting or inhibiting activity of a dendrite at another neuron.

Neuron *fires* when exciting activity surpasses inhibitory activity

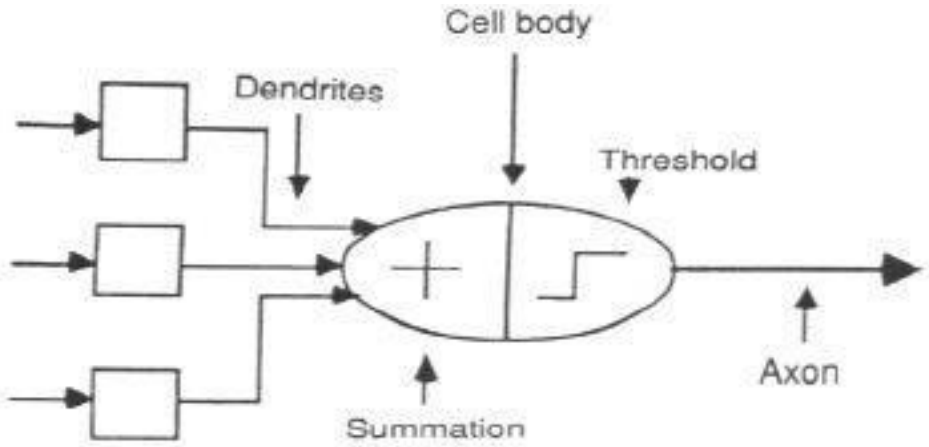
Learning changes the effectiveness of the synapses

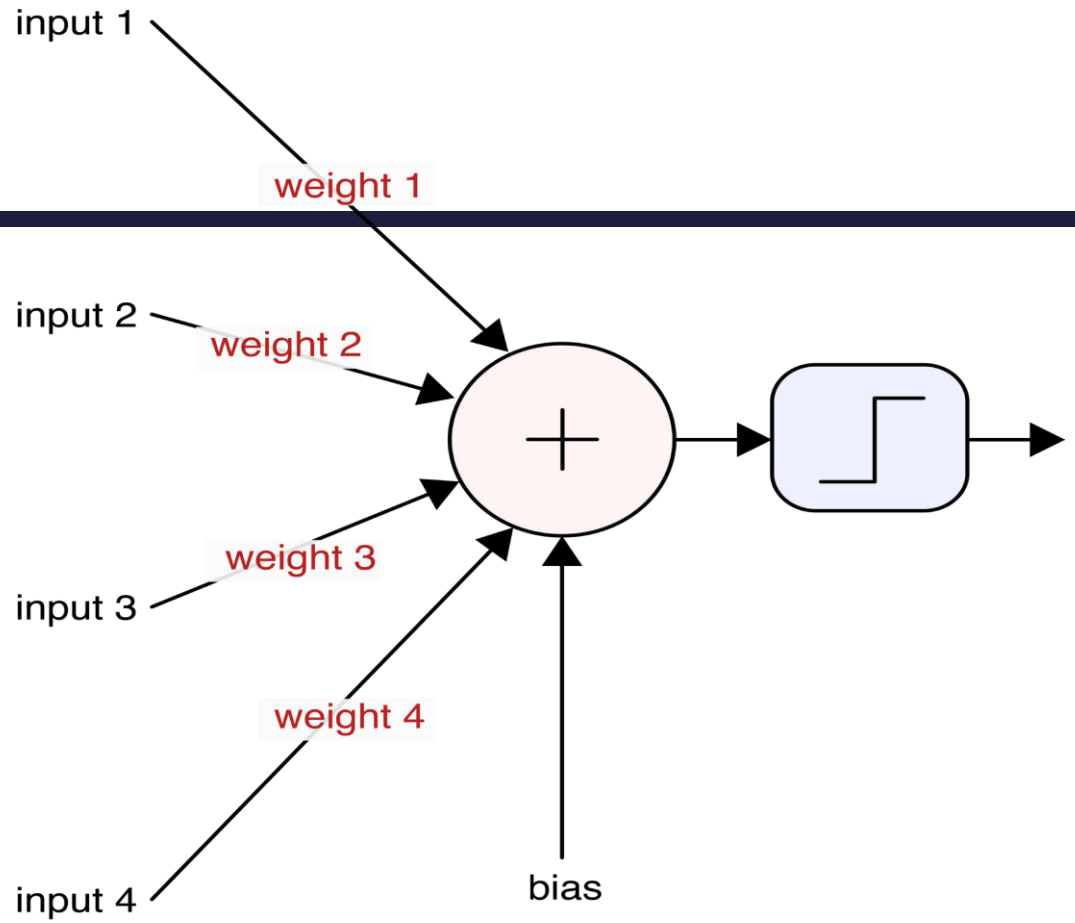
Neuron Model

Natural neurons



Abstract neuron model



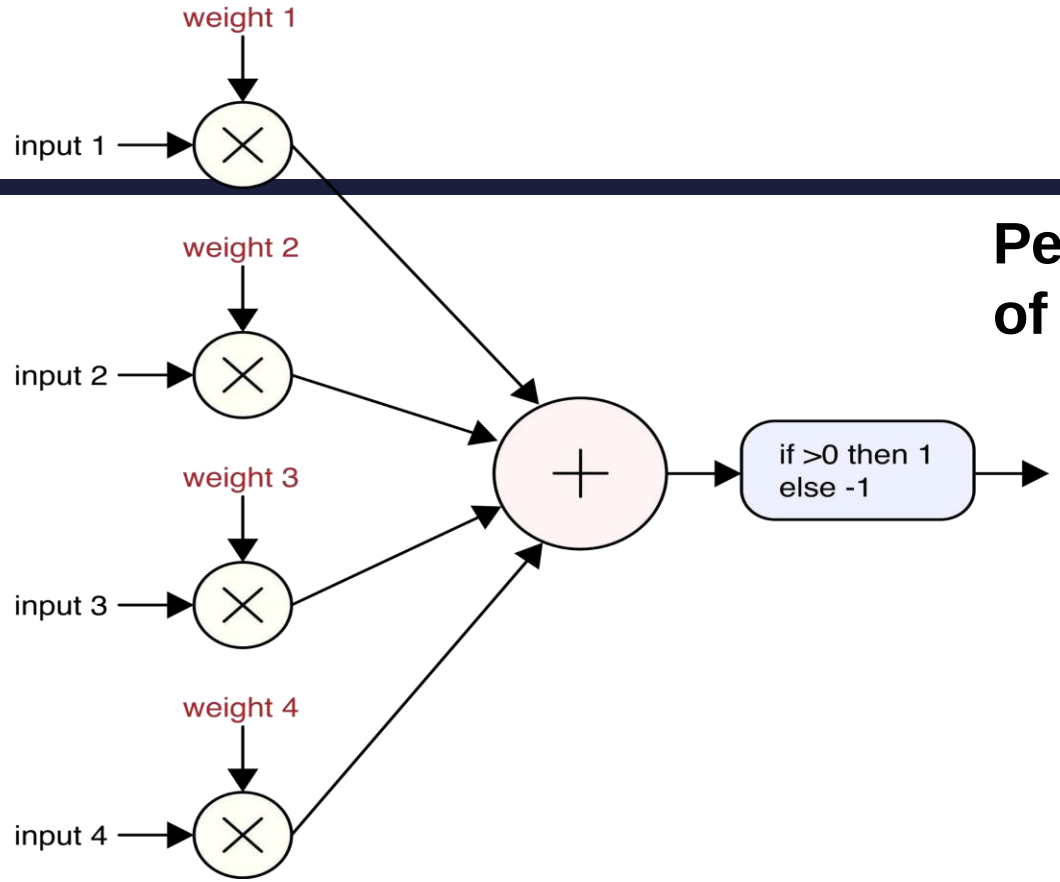


Note:

- In neural network diagrams, weights and the nodes where they multiply are not drawn
- Weights are always there and they always modifies the input . They're just not drawn.
- Output of the activation function might take on any value.
- Variety of activation functions each with its own pros and cons.

A neuron is often drawn with the weights on the arrows. This "implicit multiplication" is common in machine learning figures. We've also replaced the threshold function with a step, to remind us that any activation function can follow the sum

The "bias trick" in action. Rather than show the bias term explicitly. We pretend it's another input with its own weight.



Perceptron mimics the functional behaviour of neuron

A schematic view of a perceptron. Each input is a single number, and it's multiplied by a corresponding real number called its weight.

The results are all added together, and then tested against a threshold.

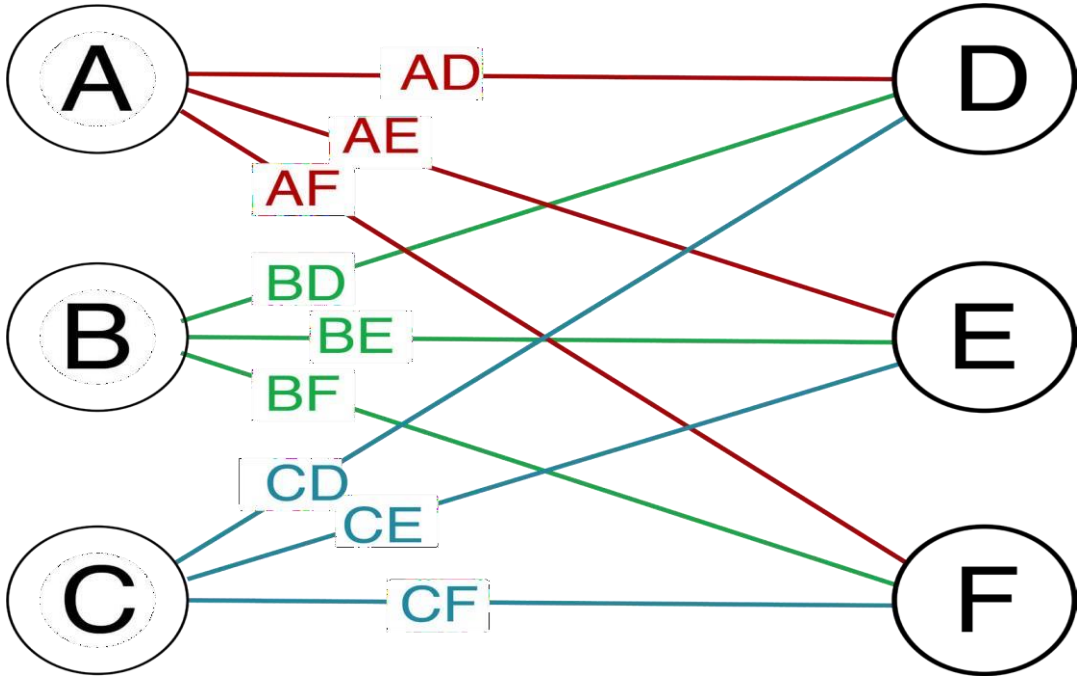
If results are positive, perceptron outputs +1, else −1.

Modern Artificial Neurons

- Modern Neurons are only slightly generalized from the original perceptrons.
- Still called “perceptrons” or “neurons”
- Two changes to original perceptron: one at the input, and one at the output.

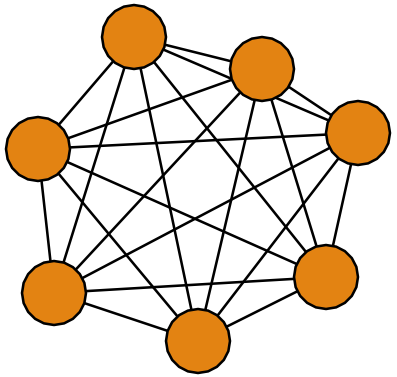
  1. Input - Provide each neuron with one more input, called bias. It’s a number that is directly added into the sum of all the weighted inputs. Each neuron has its own bias.

  2. Output - Replace threshold with an activation function, i.e., a mathematical function that takes the sum (including the bias) as input and returns a new floating-point value as output.

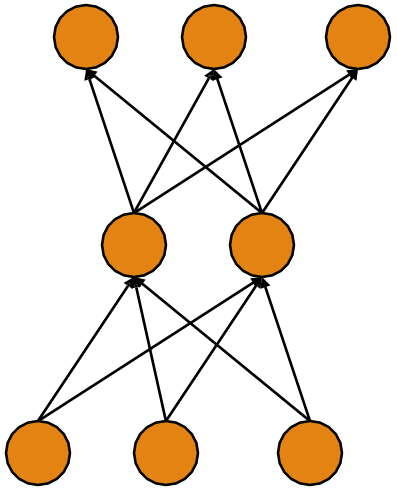


The weights are shown explicitly in this diagram. Following convention, each weight is given a two-letter name formed by combining the names of the neuron that produced the output on that weight's wire, with the name of the neuron that receives the value as input. For example, BF is the weight that multiplies the output of B for use by F

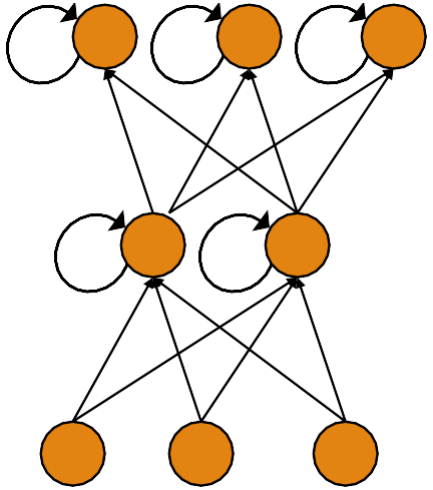
Topologies of Neural Networks



completely connected

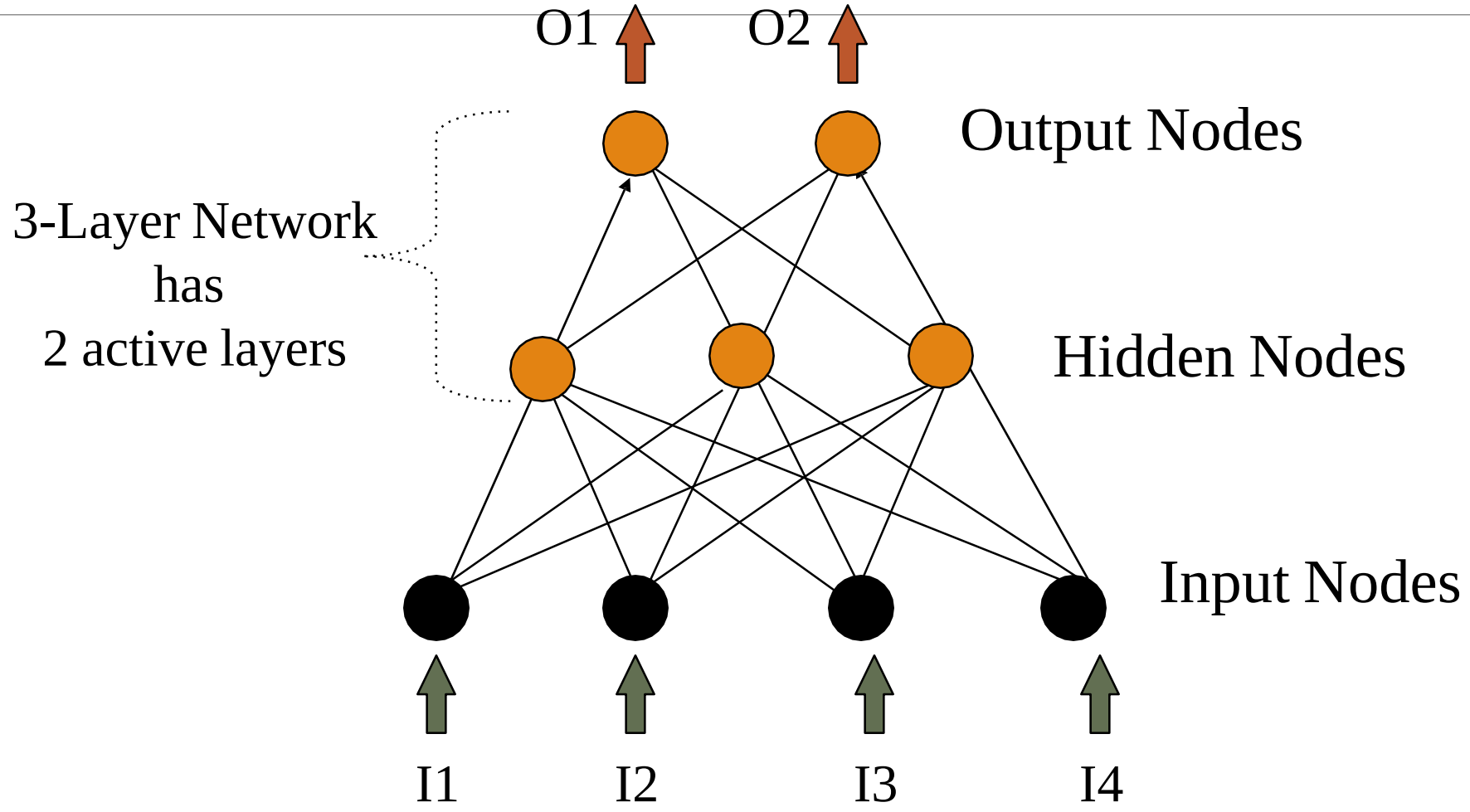


feedforward (directed, a-cyclic)

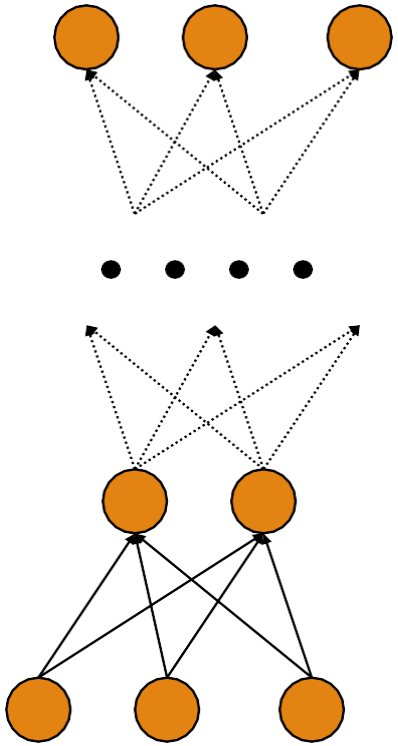


recurrent (feedback connections)

Artificial Neurons



Multilayer Perceptron



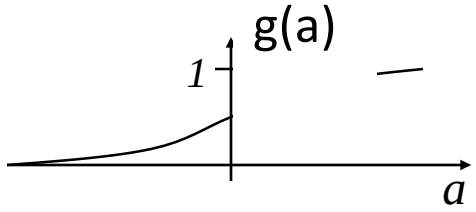
Output neurons

One or more layers of hidden units (hidden layers)

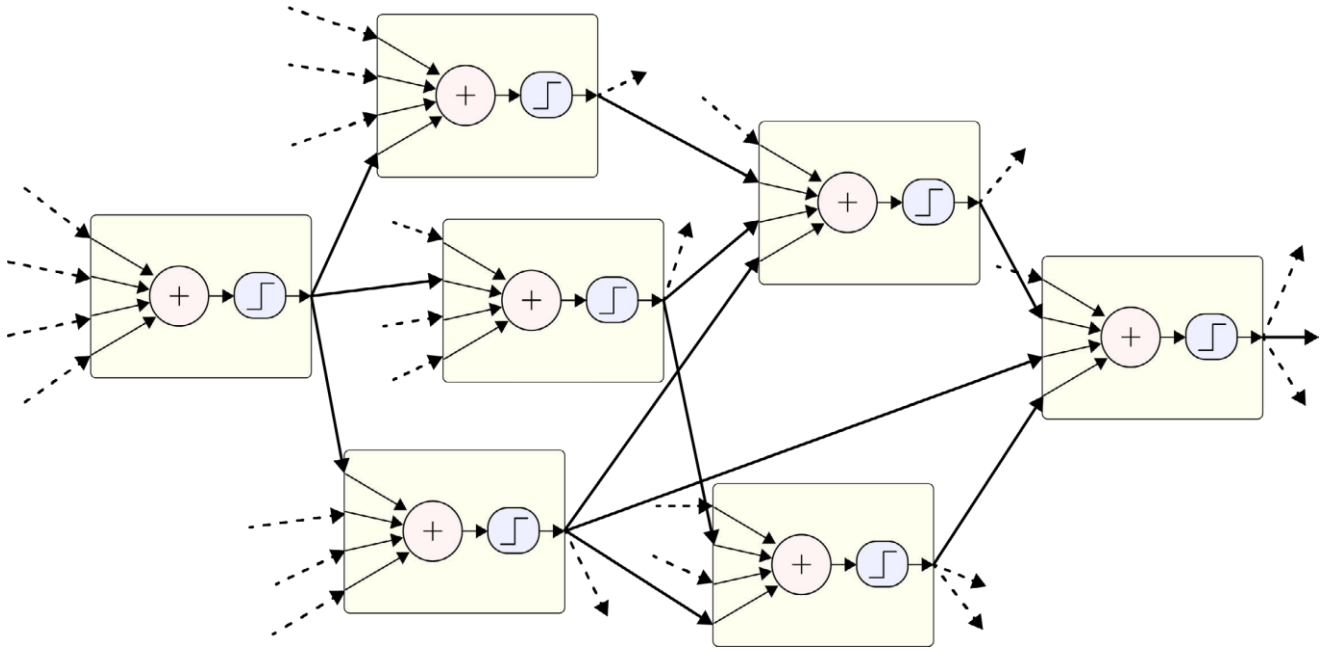
Input nodes

The most common output function (Sigmoid):

$$g(a) = \frac{1}{1 + e^{-\beta a}}$$



(non-linear squashing function)



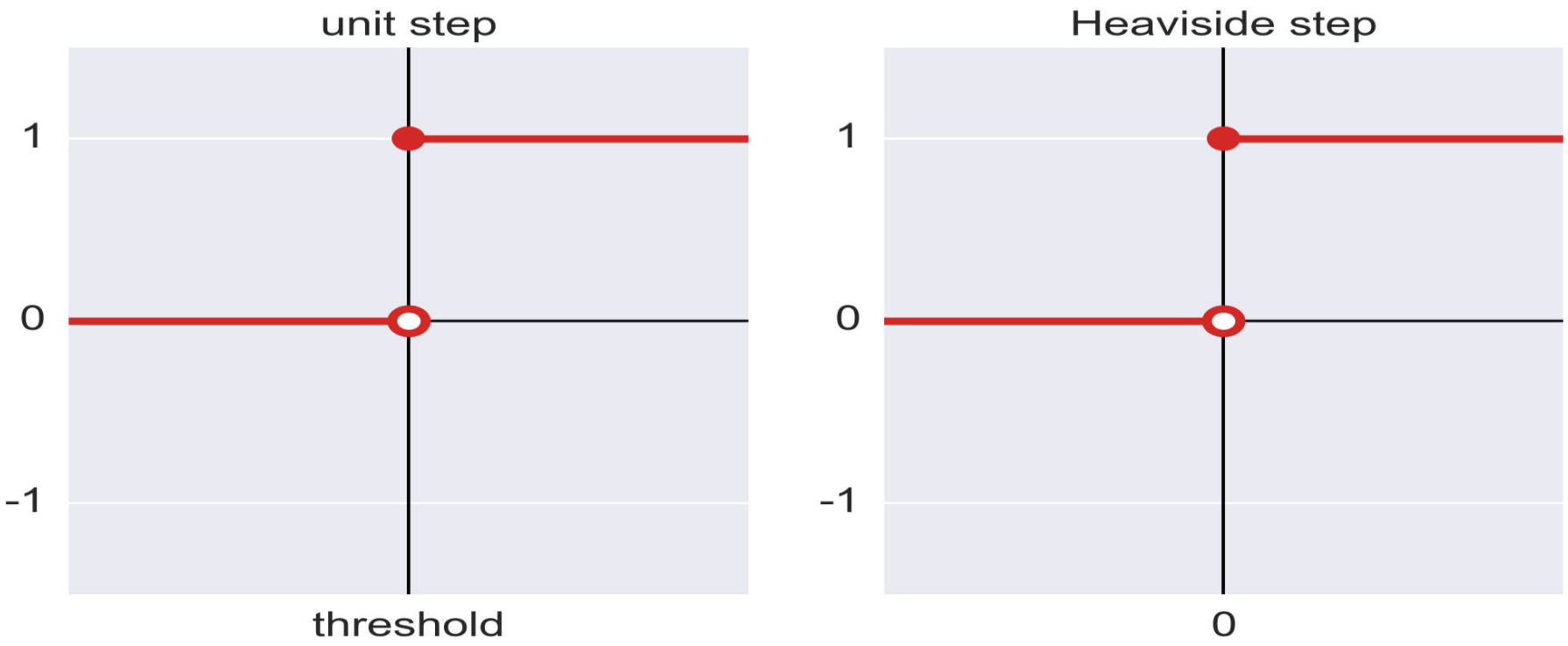
A piece of a larger network of artificial neurons. Each neuron receives its inputs from other neurons The dashed lines show connections coming from outside this little cluster.

Activation Functions

The last step in an artificial neuron is to apply an activation function to the value it computes.

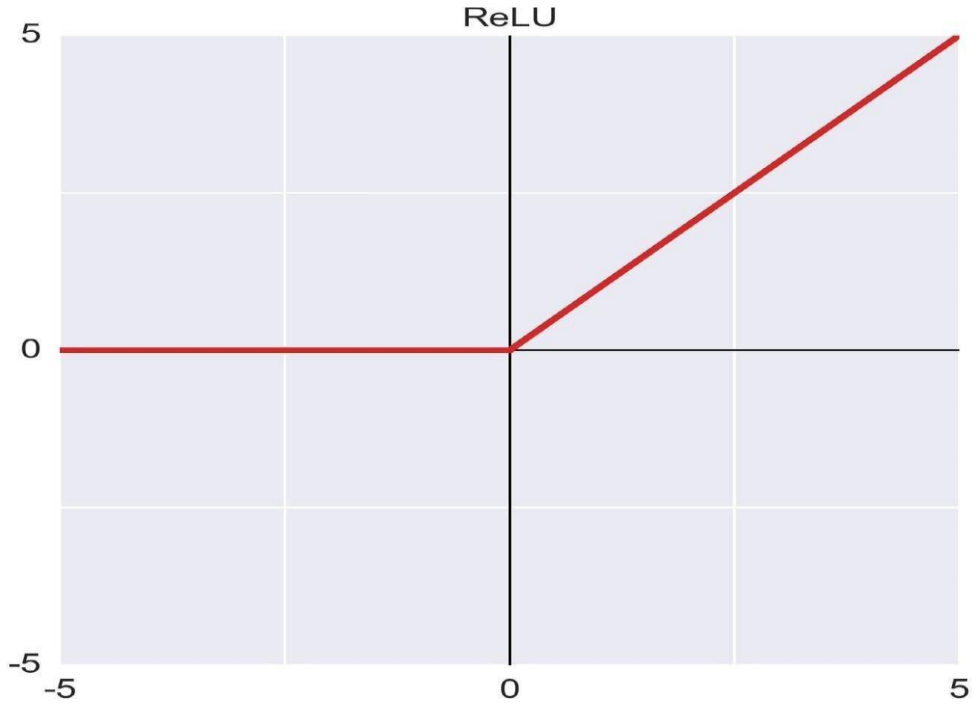
Variety of popular activation functions used today.

Activation function – Step



A couple of popular step functions.
Left: Unit step has a value of 0 to left of the threshold, and 1 to the right.
Right: The Heaviside step is a unit step where the threshold is 0.

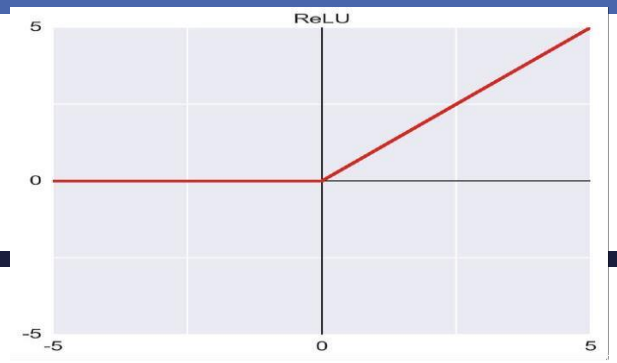
Activation function –ReLU

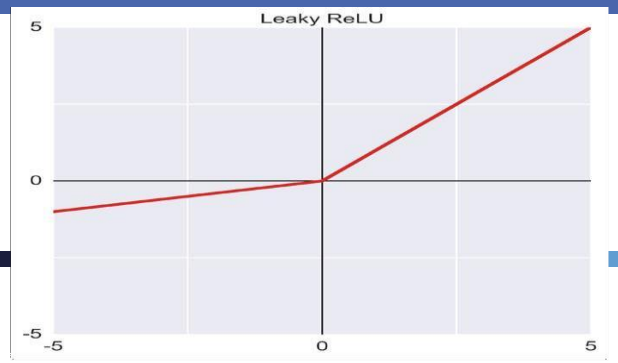


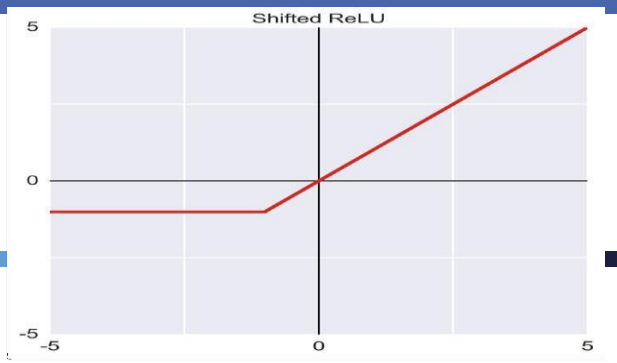
- ReLU is not a linear function.
- ReLU is popular because it's a simple and fast way to include a non-linearity in our artificial neurons.
- ReLU performs well and is often the first choice of activation function when building a new network.
- There are good mathematical reasons to use ReLU

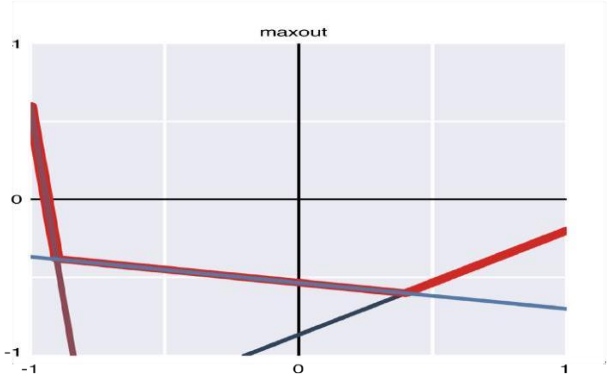
- The ReLU, or rectified linear unit.
- Output is 0 for all negative inputs. Else, output = input

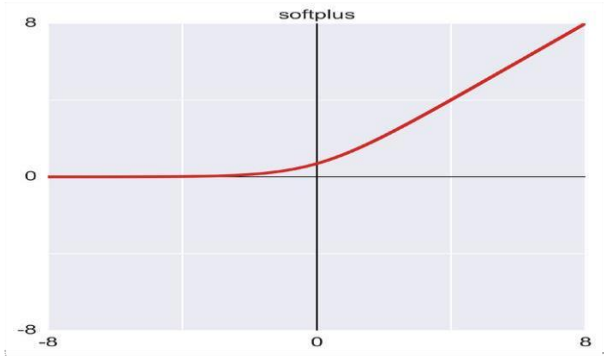
Functions

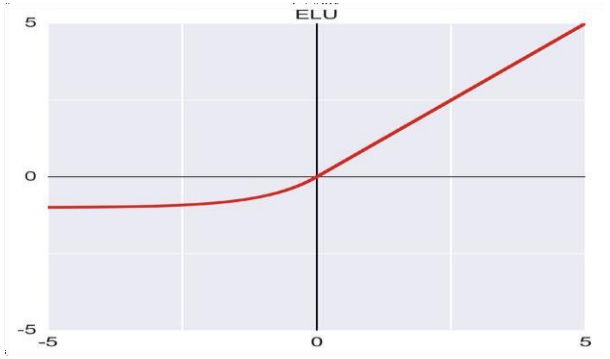


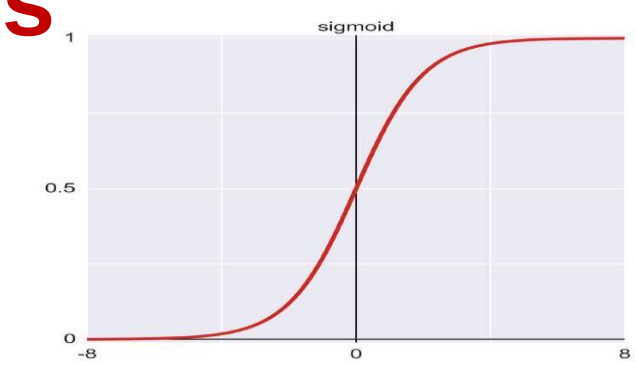


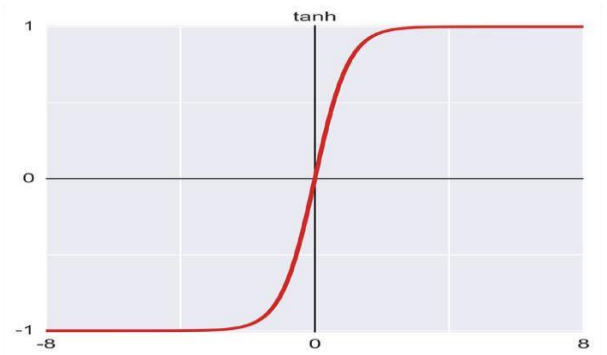


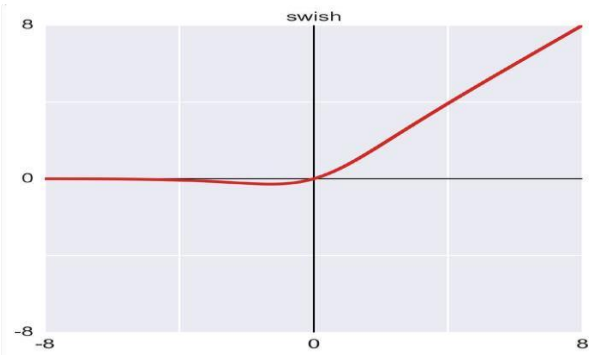












- Top row, left to right: ReLU, leaky ReLU, shifted ReLU.
- Middle row: maxout, softplus, ELU.
- Bottom row: sigmoid, tanh, swish.

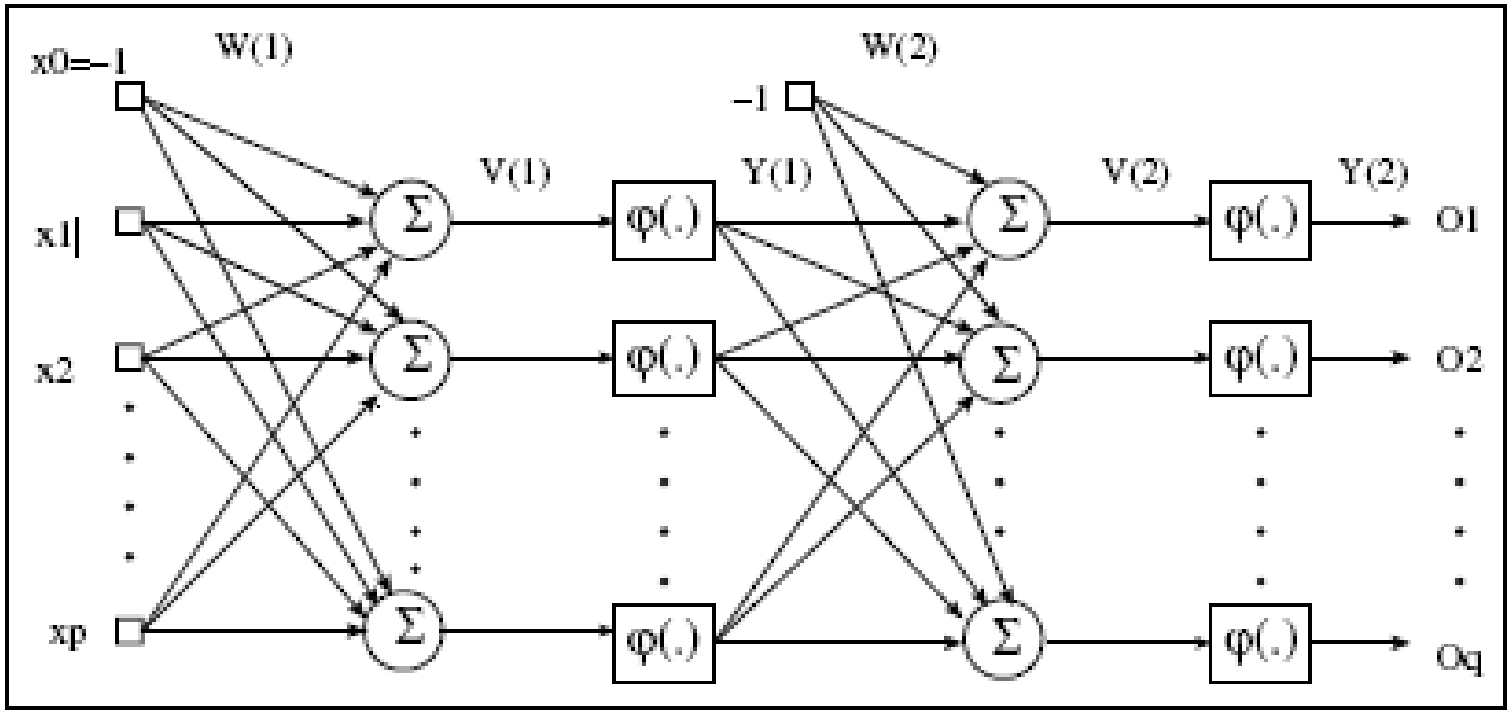
Learning in a ANN

Perceptron Learning Algorithm:

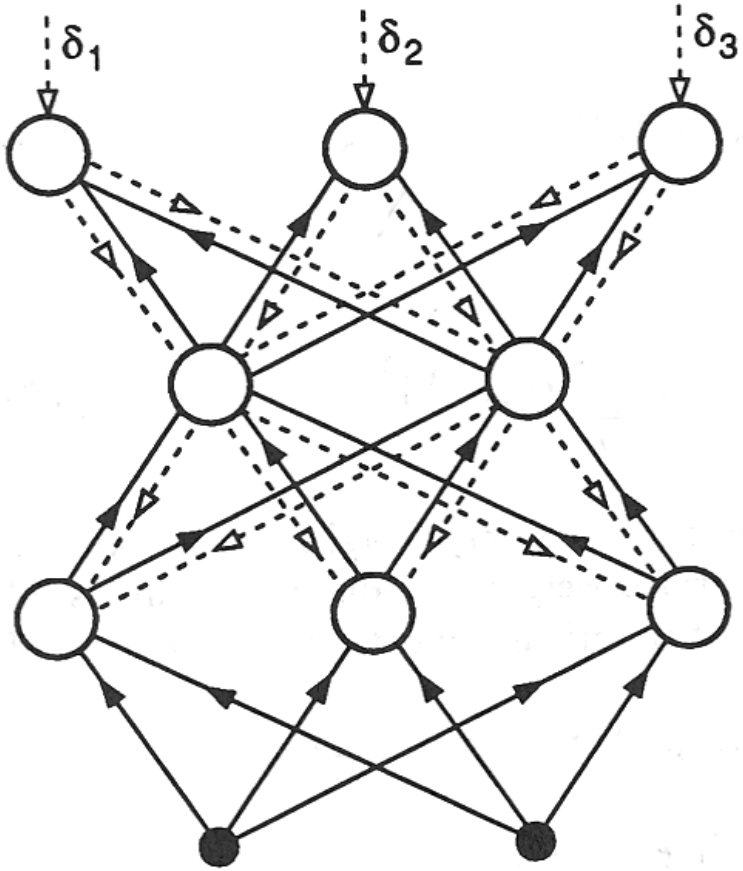
1. Initialize weights
2. Present a pattern and target output
3. Compute output :
4. Update weights :

Repeat starting at 2 until acceptable level of error

Forward Propagation



Back propagation



ANN Forward Propagation

Bias Nodes

- Add one node to each layer that has constant output

Forward propagation

- Calculate from input layer to output layer
- For each neuron:
  - Calculate weighted average of input
  - Calculate activation function

ANN Forward Propagation

Apply input vector **X** to layer of neurons.

Calculate

$$V_j(n) = \sum_{i=1}^{p} (W_{ji} X_i + \text{Bias})$$

- where X_i is the activation of previous layer neuron i
- W_{ji} is the weight of going from node i to node j
- p is the number of neurons in the previous layer

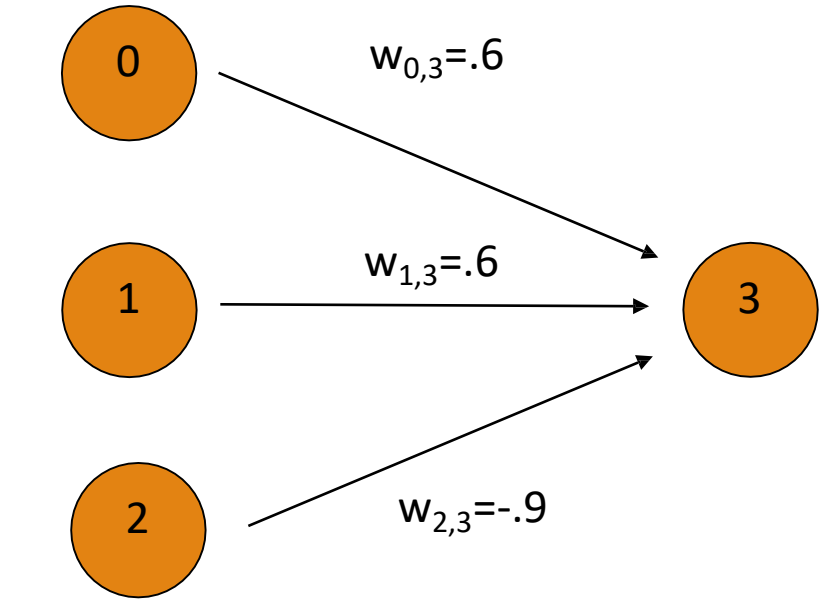
Calculate output activation

$$Y_j(n) = \frac{1}{1 + \exp(-V_j(n))}$$

ANN Forward Propagation

Example 1: ADALINE Neural Network

- Calculates and of inputs

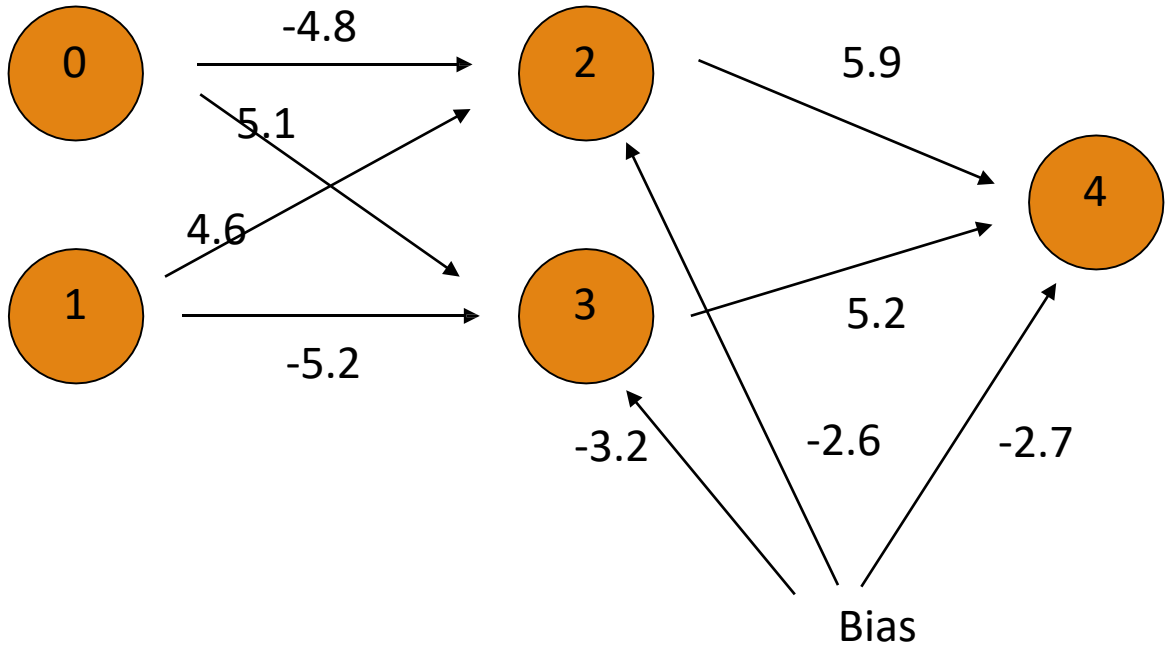


threshold function is step function

ANN Forward Propagation

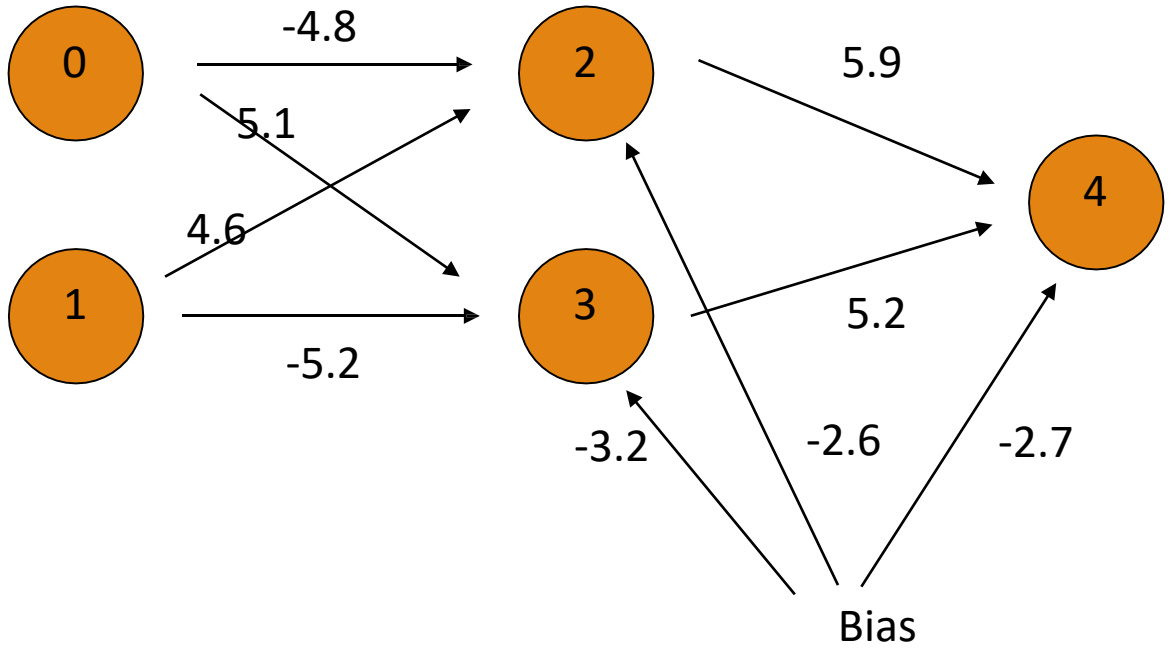
Example 2: Three layer network

- Calculates xor of inputs



ANN Forward Propagation

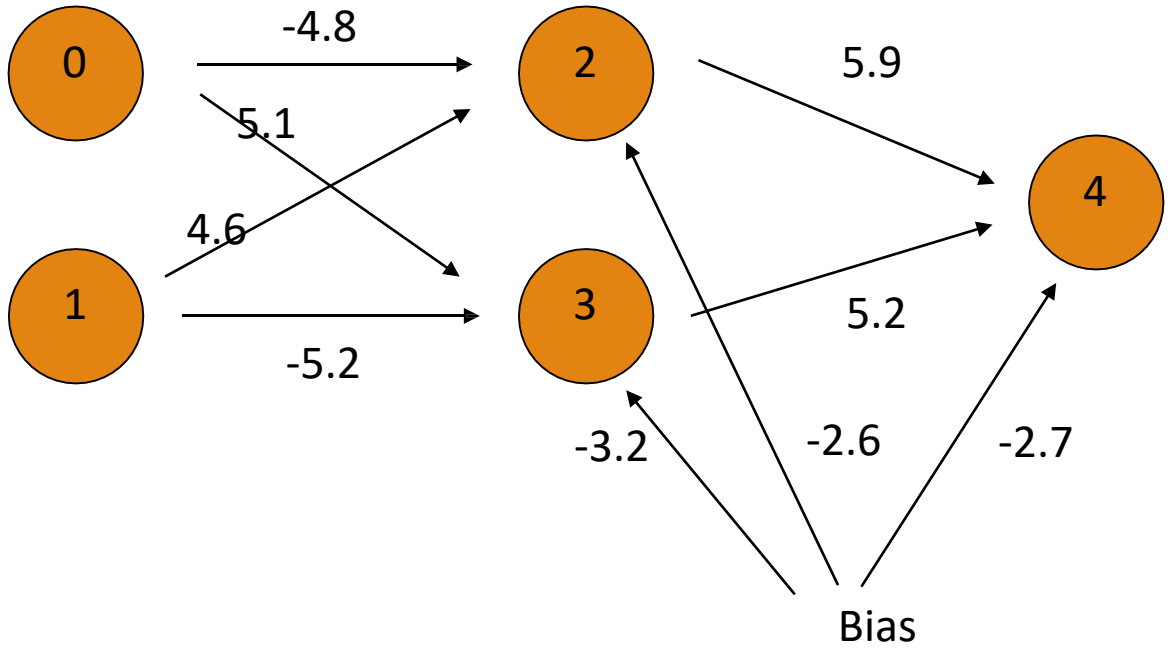
Input (0,0)



ANN Forward Propagation

Input (0,0)

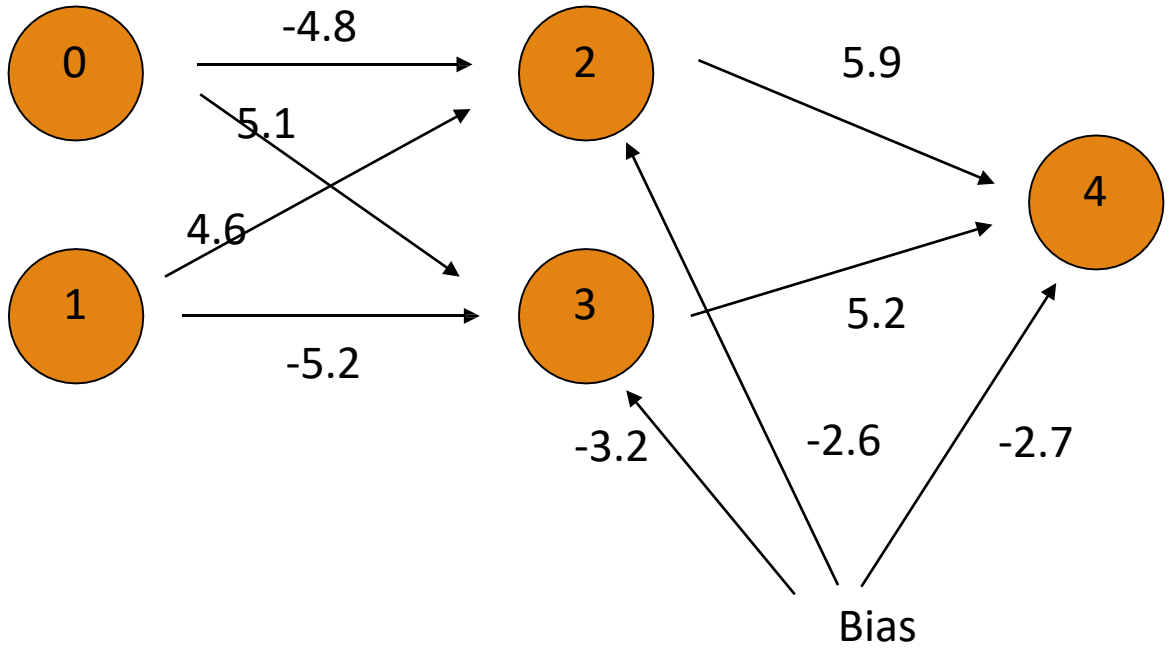
- Node 2 activation is $\varphi(-4.8 \times 0 + 4.6 \times 0 - 2.6) = 0.0691$



ANN Forward Propagation

Input (0,0)

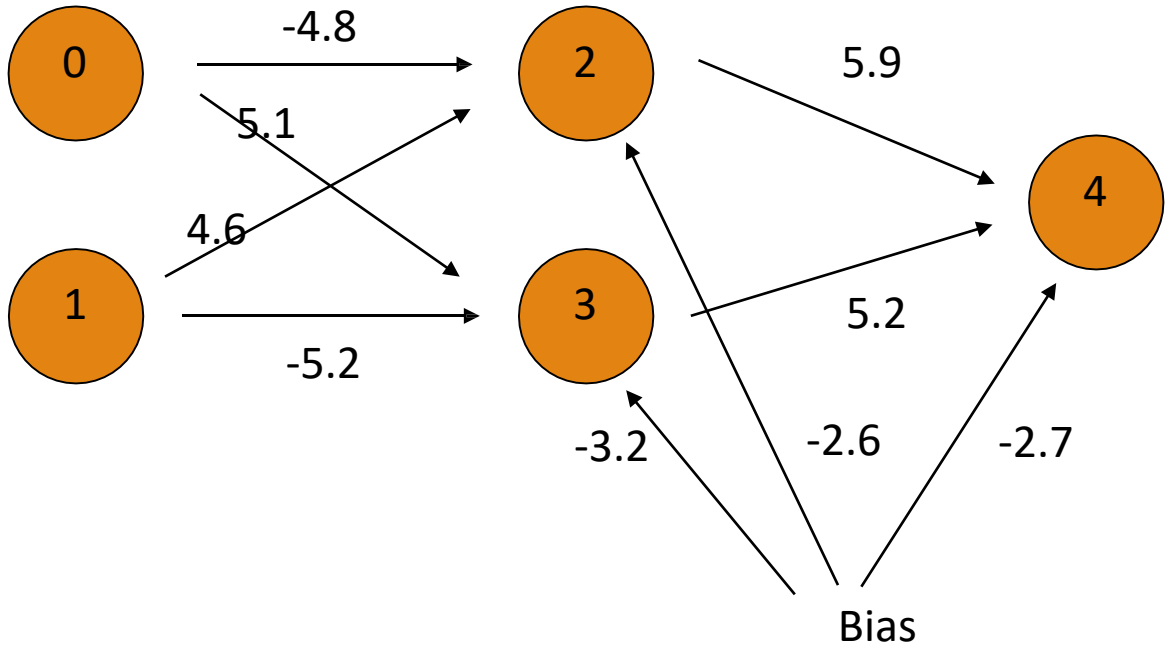
- Node 3 activation is $\varphi(5.1 \times 0 - 5.2 \times 0 - 3.2) = 0.0392$



ANN Forward Propagation

Input (0,0)

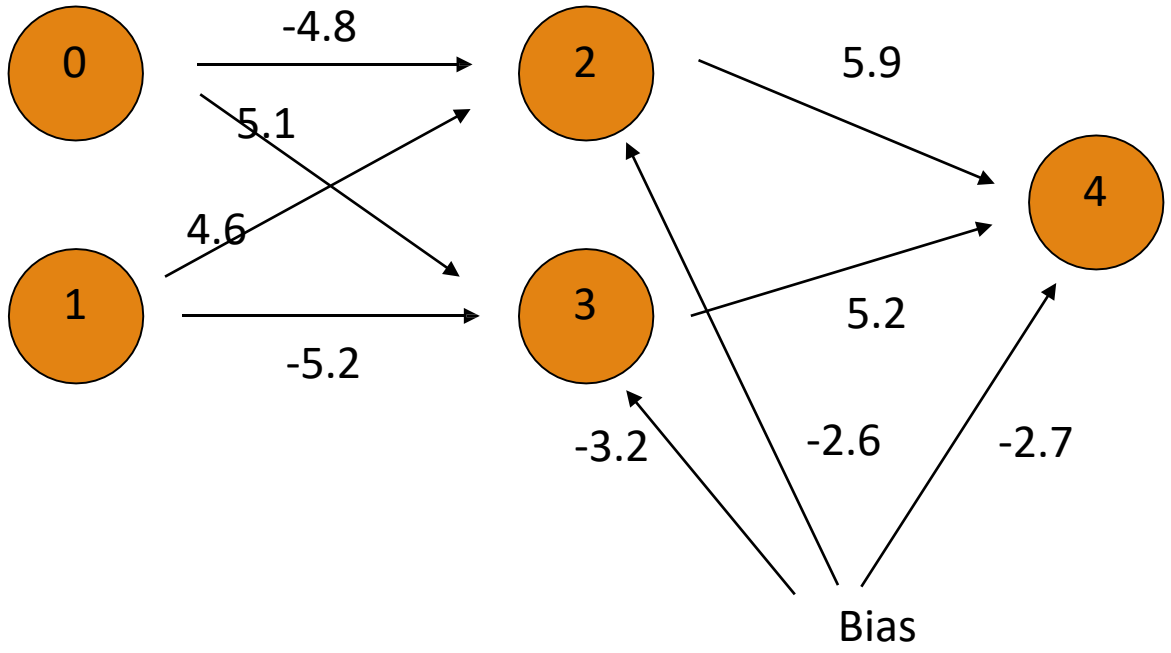
- Node 4 activation is $\varphi(5.9 \times 0.069 + 5.2 \times 0.069 - 2.7) = 0.110227$



ANN Forward Propagation

Input (0,1)

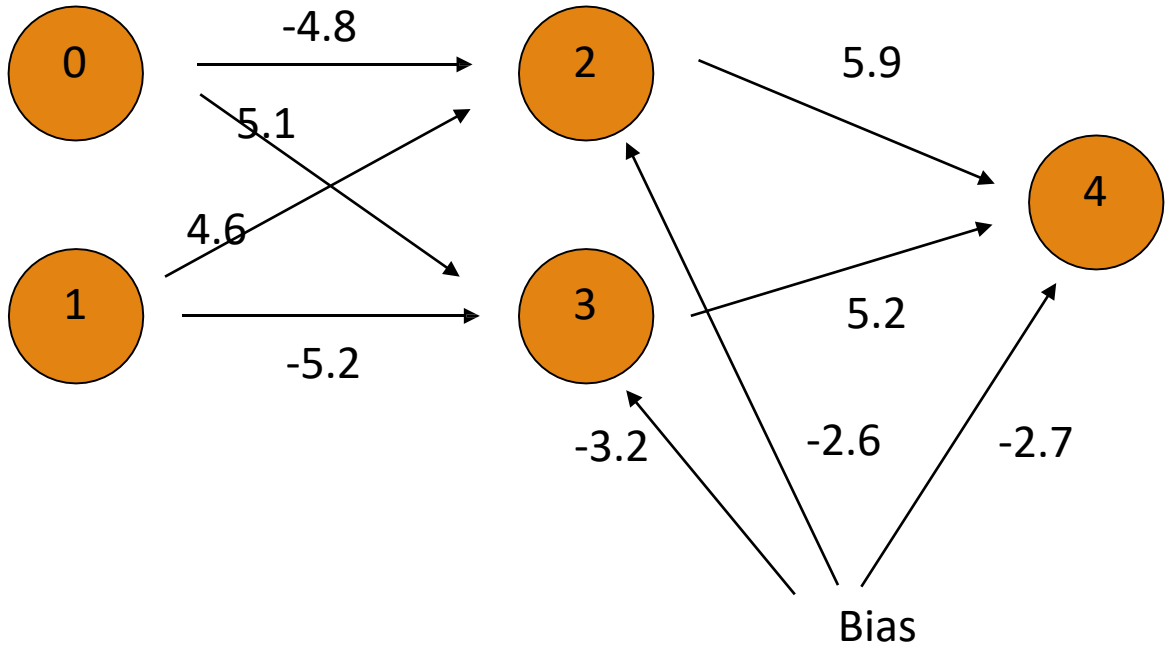
- Node 2 activation is $\varphi(4.6 \cdot -2.6) = 0.153269$



ANN Forward Propagation

Input (0,1)

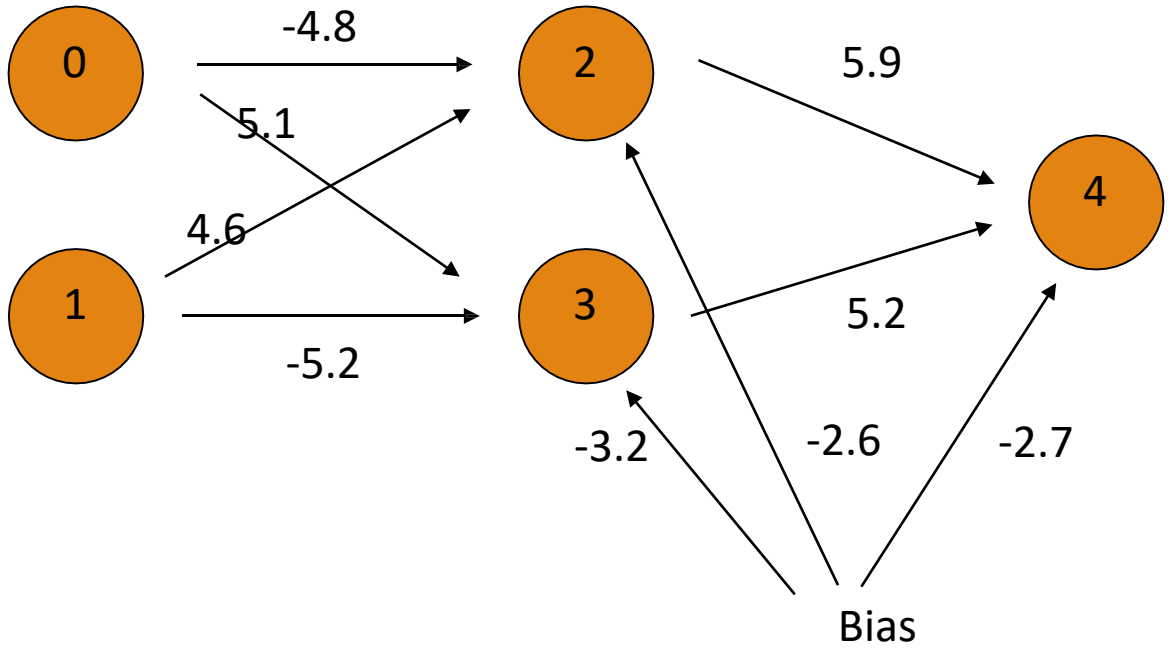
- Node 3 activation is $\varphi(-5.2\ -3.2) = 0.000224817$



ANN Forward Propagation

Input (0,1)

- Node 4 activation is $\varphi(5.9 \times 0.153269 + 5.2 \times 0.000224817 - 2.7) = 0.923992$



Neuron Model

- Perceptron rule
  - Calculate weighted average of input

$$V_j(n) = \sum_{i=1}^{p} (W_{ji} X_i + \text{Bias})$$

  - Output activation level is

$$Y_j(n) = \frac{1}{1 + \exp(-V_j(n))}$$

Firing Rules:

- Threshold rules:
  - Calculate weighted average of input
  - Fire if larger than threshold

$$\phi(\nu) = \begin{cases} 1 & \nu \geq \frac{1}{2} \\ \nu & 0 \leq \nu \leq \frac{1}{2} \\ 0 & \nu \leq 0 \end{cases}$$

Note: Network can learn a non-linearly separated set of outputs. Need to map output (real value) into binary values.

Neuron Model

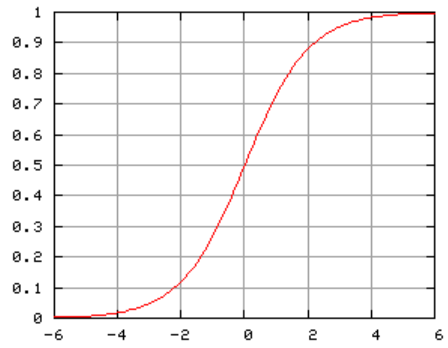
Firing Rules: Sigmoid functions:

- Hyperbolic tangent function

$$\varphi(v) = \tanh(v/2) = \frac{1-\exp(-v)}{1+\exp(-v)}$$

- Logistic activation function

$$\varphi(v) = \frac{1}{1+\exp(-v)}$$



ANN Training

Weights are determined by *training*

- Back-propagation:
  - On given input, compare actual output to desired output.
  - Adjust weights to output nodes.
  - Work backwards through the various layers
- Start out with initial random weights
  - Best to keep weights close to zero (<<10)

ANN Training

Weights are determined by *training*

- Need a training set
  - Should be representative of the problem
- During each training epoch:
  - Submit training set element as input
  - Calculate the error for the output neurons
  - Calculate average error during epoch
  - Adjust weights

ANN Training

Error is the mean square of differences in output layer

$$E(\vec{x}) = \frac{1}{2}\sum_{k=1}^{K}(y_k(\vec{x}) - t_k(\vec{x}))^2$$

y – observed output

t – target output

Note: Error of training epoch is the average of all errors.

ANN Training

Update weights and thresholds using

- Weights

$$w_{j,k} = w_{j,k} + (-\eta)\frac{\partial E(\vec{x})}{\partial w_{jk}}$$

- Bias

$$\theta_k = \theta_k + (-\eta)\frac{\partial E(\vec{x})}{\partial \theta_k}$$

- η is a possibly time-dependent factor that should prevent overcorrection

ANN Training

Using a sigmoid function, we get

$$\frac{\partial E(\vec{x})}{\partial w_{jk}} = -y_j \delta_j$$

$$\delta_j = f'(\text{net}_j)(t_j - y_j)$$

- Logistics function φ has derivative $\varphi'(t) = \varphi(t)(1- \varphi(t))$

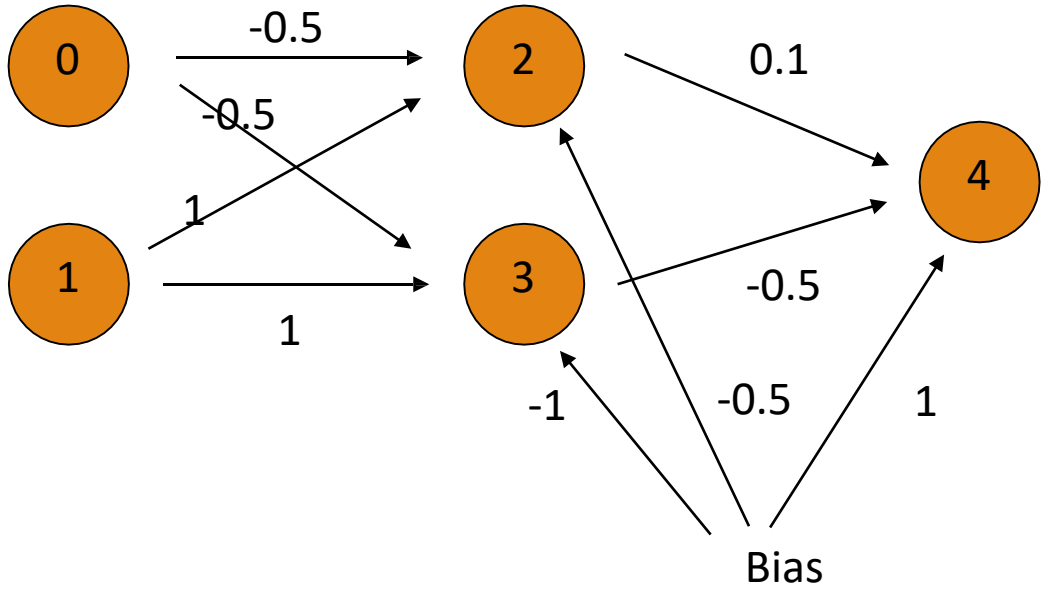
Backpropagation Computation

https://www.youtube.com/watch?v=0e0z28wAWfg

Refer attached 2 Hand written notes

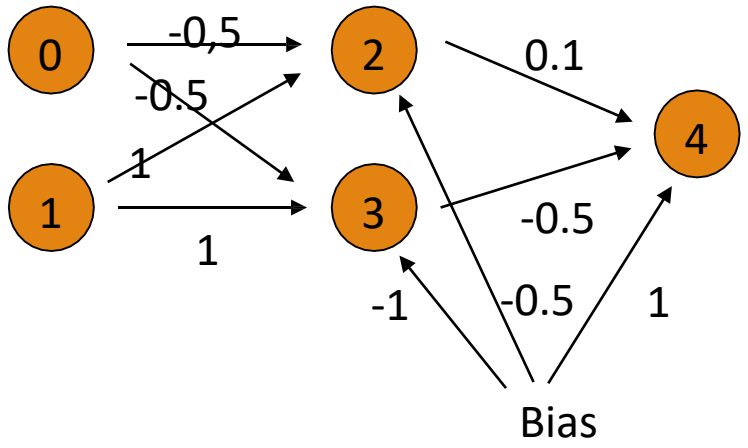
ANN Training Example

Start out with random, small weights



x1	x2	y
0	0	0.687349
0	1	0.667459
1	0	0.698070
1	1	0.676727

ANN Training Example



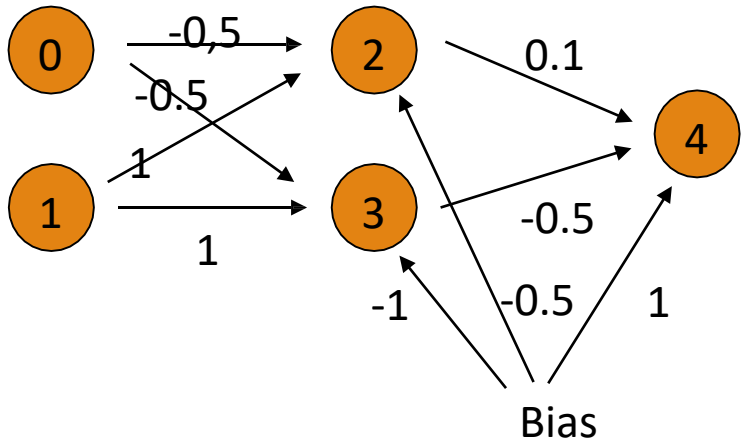
x1	x2	y	Error
0	0	0.69	0.472448
0	1	0.67	0.110583
1	0	0.70	0.0911618
1	1	0.68	0.457959

Average Error is 0.283038

ANN Training Example

Calculate the derivative of the error with respect to the weights and bias into the output layer neurons

ANN Training Example



New weights going into node 4

We do this for all training inputs, then average out the changes

net_4 is the weighted sum of input going into neuron 4:

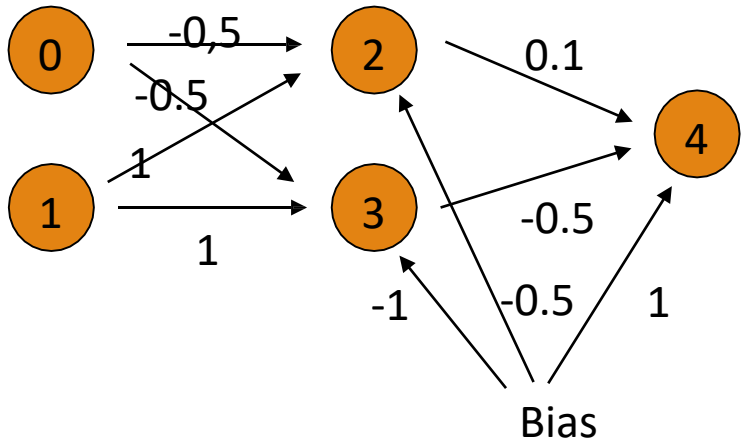
$net_4(0,0)= 0.787754$

net4(0,1)= 0.696717

net4(1,0)= 0.838124

net4(1,1)= 0.73877

ANN Training Example



New weights going into node 4

We calculate the derivative of the activation function at the point given by the net-input.

Recall the formula

$$\varphi'(t) = \varphi(t)(1- \varphi(t))$$

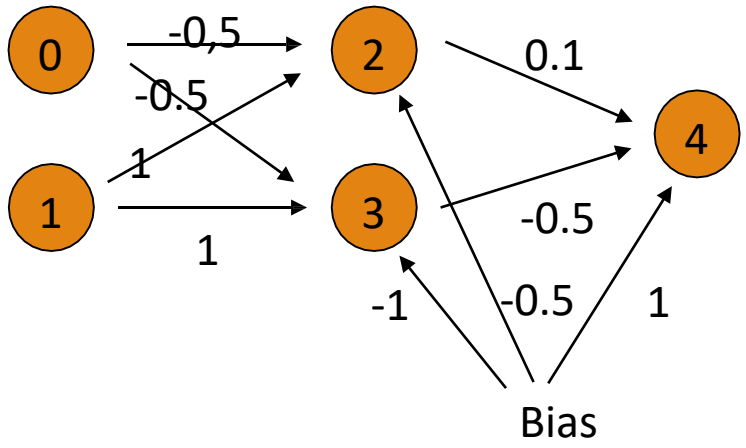
$\varphi'(\text{net}_4(0,0)) = \varphi'(0.787754) = 0.214900$

$\varphi'(\text{net}_4(0,1)) = \varphi'(0.696717) = 0.221957$

$\varphi'(\text{net}_4(1,0)) = \varphi'(0.838124) = 0.210768$

$\varphi'(\text{net}_4(1,1)) = \varphi'(0.738770) = 0.218768$

ANN Training Example



$$\frac{\partial E(\vec{x})}{\partial w_{jk}} = -y_j \delta_j$$

$$\delta_j = f'(\text{net}_j)(t_j - y_j)$$

New weights going into node 4

We now obtain δ values for each input separately:

Input 0,0:

$\delta_4 = \varphi'(\text{net}_4(0,0)) * (0 - y_4(0,0)) = -0.152928$

Input 0,1:

$\delta_4 = \varphi'(\text{net4}(0,1)) * (1 - y_4(0,1)) = 0.0682324$

Input 1,0:

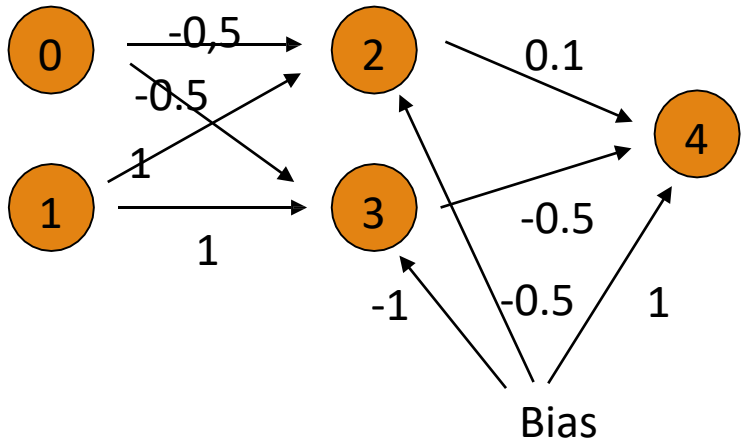
$\delta_4 = \varphi'(\text{net}_4(1,0)) * (1 - y_4(1,0)) = 0.0593889$

Input 1,1:

$\delta_4 = \varphi'(\text{net}_4(1,1)) * (0 - y_4(1,1)) = -0.153776$

Average: $\delta_4 = -0.0447706$

ANN Training Example



$$\frac{\partial E(\vec{x})}{\partial w_{jk}} = -y_j \delta_j$$

$$\delta_j = f'(\text{net}_j)(t_j - y_j)$$

New weights going into node 4

Average: δ_4 = -0.0447706

We can now update the weights going into node 4:

Let's call: E_{ji} the derivative of the error function with respect to the weight going from neuron *i* into neuron *j*.

We do this for every possible input:

$E_{4,2}$ = - output(neuron(2)* δ_4

For (0,0): $E_{4,2}$ = 0.0577366

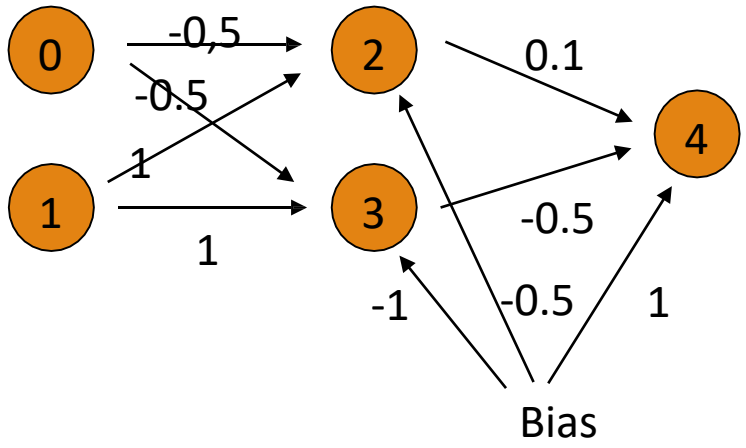
For (0,1): $E_{4,2}$ = -0.0424719

For(1,0): $E_{4,2}$ = -0.0159721

For(1,1): $E_{4,2}$ = 0.0768878

Average is 0.0190451

ANN Training Example

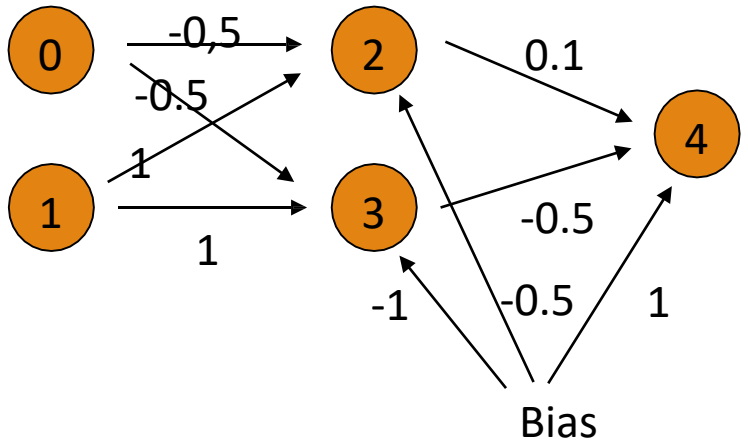


New weight from 2 to 4 is now going to be 0.1190451.

$$\frac{\partial E(\vec{x})}{\partial w_{jk}} = -y_j \delta_j$$

$$\delta_j = f'(\text{net}_j)(t_j - y_j)$$

ANN Training Example



New weights going into node 4

For (0,0): $E_{4,3}$ = 0.0411287

For (0,1): $E_{4,3}$ = -0.0341162

For(1,0): $E_{4,3}$ = -0.0108341

For(1,1): $E_{4,3}$ = 0.0580565

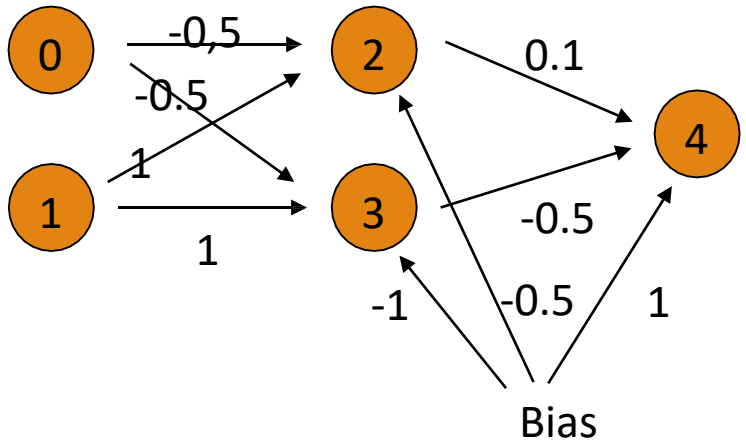
Average is 0.0135588

$$\frac{\partial E(\vec{x})}{\partial w_{jk}} = -y_j \delta_j$$

$$\delta_j = f'(\text{net}_j)(t_j - y_j)$$

New weight is -0.486441

ANN Training Example



$$\frac{\partial E(\vec{x})}{\partial w_{jk}} = -y_j \delta_j$$

$$\delta_j = f'(\text{net}_j)(t_j - y_j)$$

New weights going into node 4:

We also need to change the bias node

For (0,0): $E_{4,B}$ = 0.0411287

For (0,1): $E_{4,B}$ = -0.0341162

For(1,0): $E_{4,B}$ = -0.0108341

For(1,1): $E_{4,B}$ = 0.0580565

Average is 0.04 47706

New weight is 1.0447706

ANN Training Example

We now have adjusted all the weights into the output layer.

Next, we adjust the hidden layer

The target output is given by the delta values of the output layer

More formally:

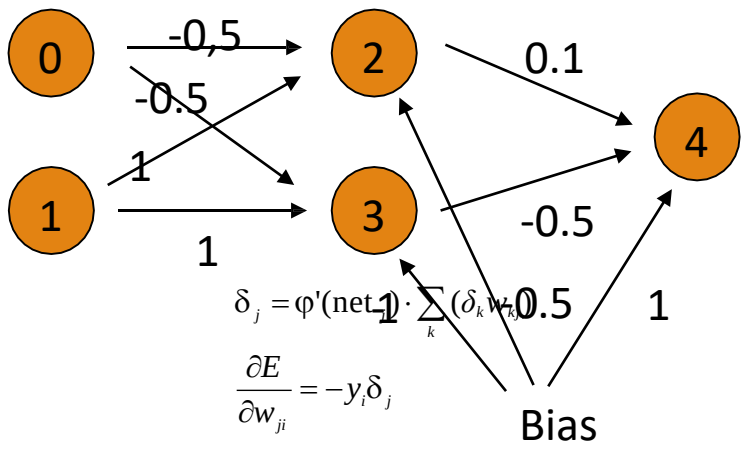
- Assume that *j* is a hidden neuron
- Assume that δ_k is the delta-value for an output neuron *k*.
- While the example has only one output neuron, most ANN have more. When we sum over *k,* this means summing over all output neurons**.**

$$\delta_j = \varphi'(\text{net}_j) \cdot \sum_k (\delta_k w_{kj})$$

- w_{kj} is the weight from neuron *j* into neuron *k*

$$\frac{\partial E}{\partial w_{ji}} = -y_i \delta_j$$

ANN Training Example



We now calculate the updates to the weights of neuron 2.

First, we calculate the net-input into 2.

This is really simple because it is just a linear functions of the arguments x_1 and x_2

$net_2 = -0.5\ x_1 + x2 - 0.5$

We obtain

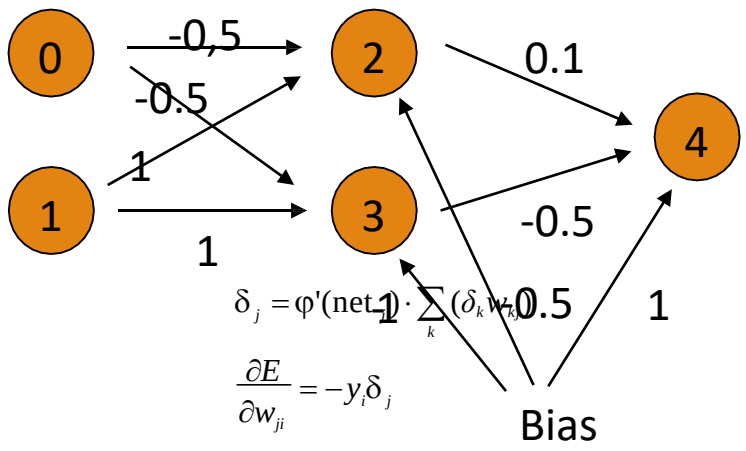
$\delta_2\ (0,0) = -\ 0.00359387$

$\delta_2(0,1) = \quad 0.00160349$

$\delta_2(1,0) = \quad 0.00116766$

$\delta_2(1,1) = \ -\ 0.00384439$

ANN Training Example



$$\delta_j = \varphi'(\text{net}_j) \cdot \sum_k (\delta_k w_{kj})$$

$$\frac{\partial E}{\partial w_{ji}} = -y_i \delta_j$$

Call E_{20} the derivative of E with respect to w_{20}.
We use the output activation for the neurons in the previous layer (which happens to be the input layer)

$E_{20}(0,0) = -\varphi(0) \cdot \delta_2(0,0) = 0.00179694$

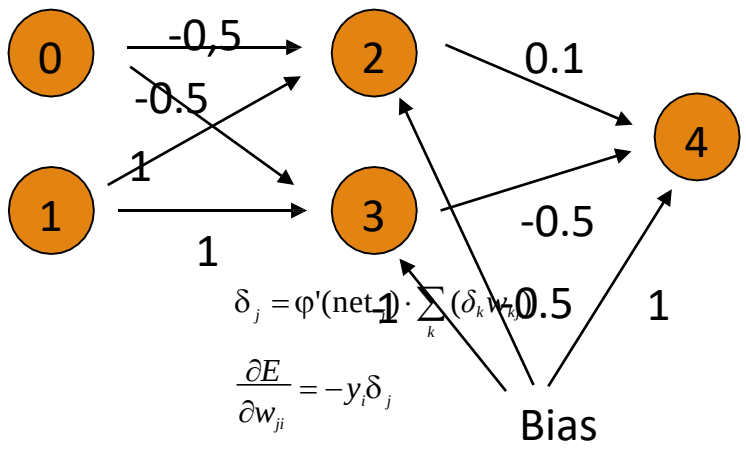
$E_{20}(0,1) = 0.00179694$

$E_{20}(1,0) = -0.000853626$

$E_{20}(1,1) = 0.00281047$

The average is 0.00073801 and the new weight is -0.499262

ANN Training Example



Call E_{21} the derivative of E with respect to w_{21}.
We use the output activation for the neurons in the previous layer (which happens to be the input layer)

$E_{21}(0,0) = -\varphi(1)\cdot\delta_2(0,0) =$ 0.00179694

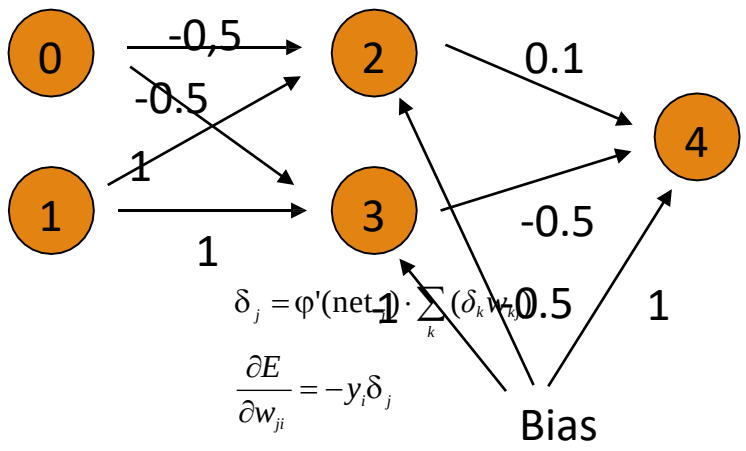
$E_{21}(0,1) =$ -0.00117224

$E_{21}(1,0) =$ -0.000583829

$E_{21}(1,1) =$ 0.00281047

The average is 0.000712835 and the new weight is 1.00071

ANN Training Example



Call E_{2B} the derivative of E with respect to w_{2B}.
Bias output is always -0.5

$E_{2B}(0,0) = - -0.5 \cdot \delta_2(0,0) =$ 0.00179694

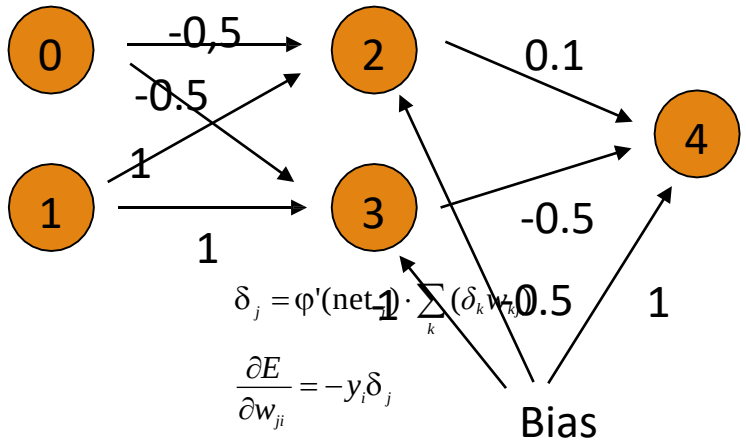
$E_{2B}(0,1) =$ -0.00117224

$E_{2B}(1,0) =$ -0.000583829

$E_{2B}(1,1) =$ 0.00281047

The average is 0.00058339 and the new weight is -0.499417

ANN Training Example



$$\delta_j = \varphi'(\text{net}_j) \cdot \sum_k (\delta_k w_{kj})$$

$$\frac{\partial E}{\partial w_{ji}} = -y_i \delta_j$$

Similarly calculate the updates to the weights of neuron 3.

…

ANN Training

Need to measure effectiveness of training

- Need training sets
- Need test sets.

There can be no interaction between test sets and training sets.

- Example of a Mistake:
  - Train ANN on training set.
  - Test ANN on test set.
  - Results are poor.
  - Go back to training ANN.
- After this, there is no assurance that ANN will work well in practice.
  - In a subtle way, the test set has become part of the training set.

ANN Training

Convergence

- ANN back propagation uses gradient decent.
- Naïve implementations can
  - overcorrect weights
  - undercorrect weights
- In either case, convergence can be poor

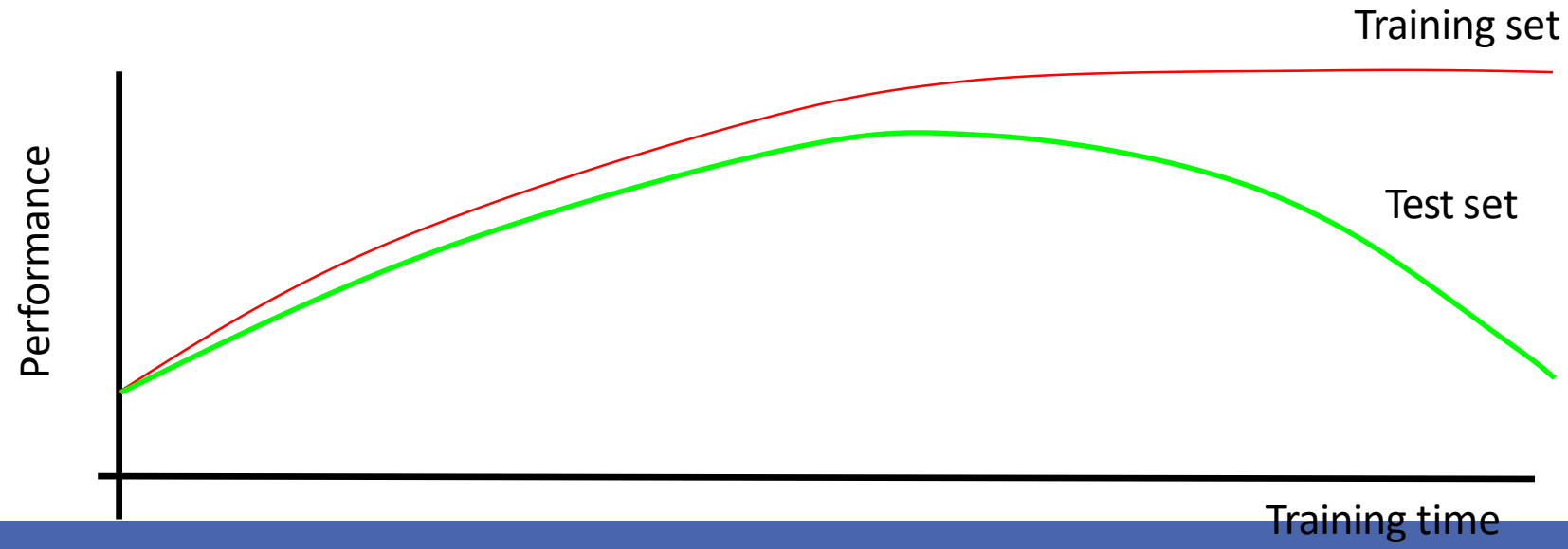
Stuck in the wrong place

- ANN starts with random weights and improves them
- If improvement stops, we stop algorithm
- No guarantee that we found the best set of weights
- Could be stuck in a local minimum

ANN Training

Overtraining

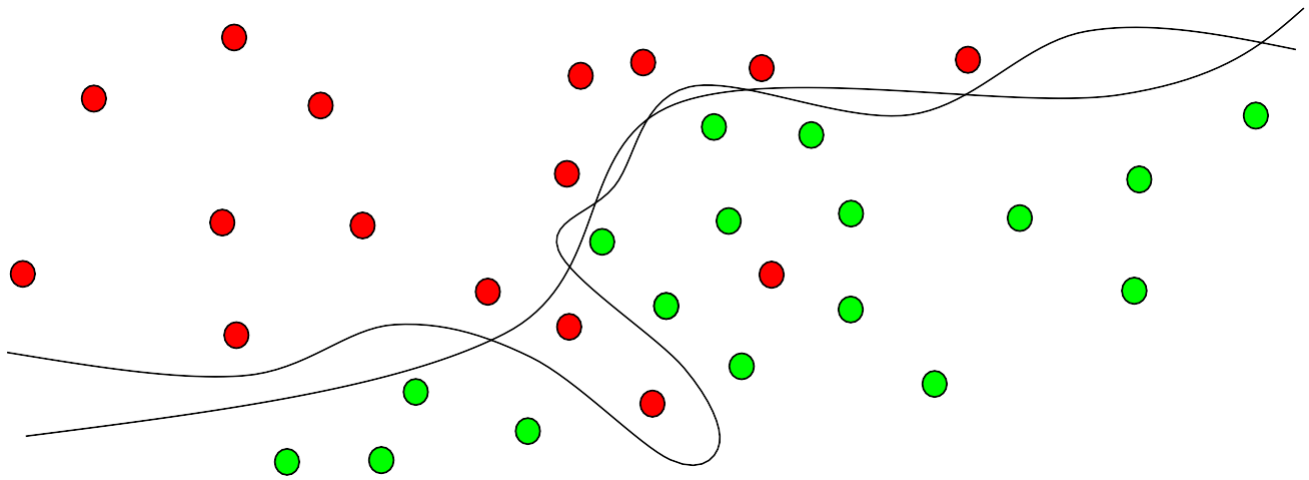
- An ANN can be made to work too well on a training set
- But loose performance on test sets



ANN Training

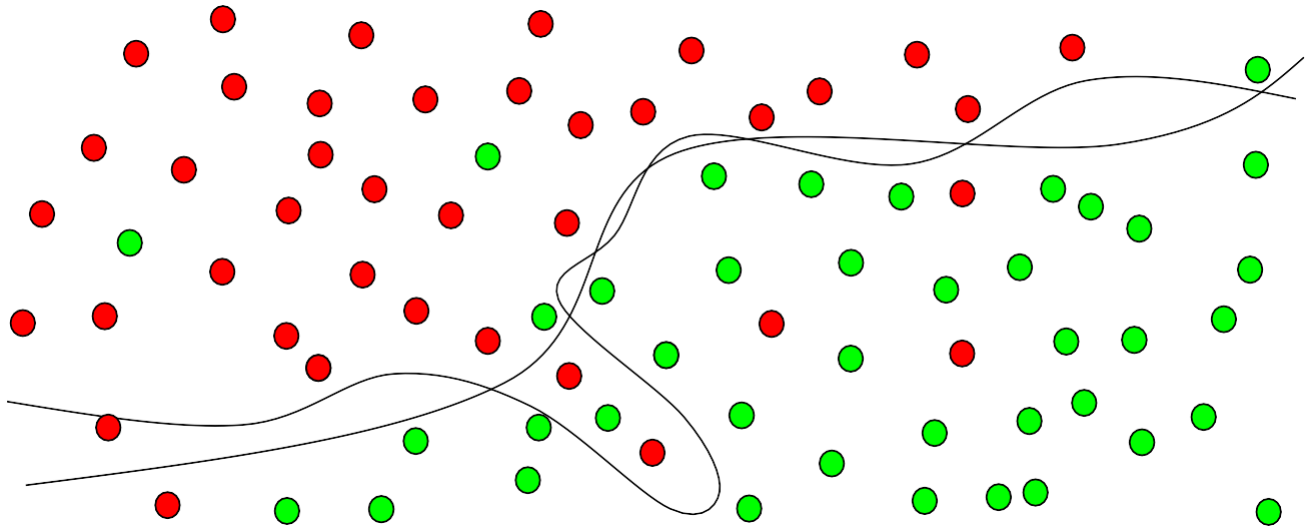
Overtraining

- Assume we want to separate the red from the green dots.
- Eventually, the network will learn to do well in the training case
- But have learnt only the particularities of our training set



ANN Training

Overtraining



ANN Training

Improving Convergence

- Many Operations Research Tools apply
  - Simulated annealing
  - Sophisticated gradient descent

ANN – Issues

Number of layers

- Apparently, three layers is almost always good enough and better than four layers.
- Also: fewer layers are faster in execution and training

How many hidden nodes?

- Many hidden nodes allow to learn more complicated patterns
- Because of overtraining, almost always best to set the number of hidden nodes too low and then increase their numbers.

ANN - Issues

Interpreting Output

- ANN's output neurons do not give binary values.
  - Good or bad
  - Need to define what is an accept.
- Can indicate *n* degrees of certainty with *n*-1 output neurons.
  - Number of firing output neurons is degree of certainty

Applications

Pattern recognition

- Network attacks
- Breast cancer
- Object and face detection
- handwriting recognition

Auto-association

- ANN trained to reproduce input as output
  - Noise reduction
  - Compression
  - Finding anomalies

Refer Text Book & Class notes for

Deep Networks: Deep Feedforward Networks - Learning XOR - Gradient Based learning - Hidden Units - Back-propagation and other Differential Algorithms - Regularization for Deep Learning

Book

Ian Goodfellow, Yoshua Bengio, Aaron Courville, “Deep Learning”, The MIT Press, 2016.