Department of
Mathematics &
Computer Science

18. Learning From Examples

What is Learning?

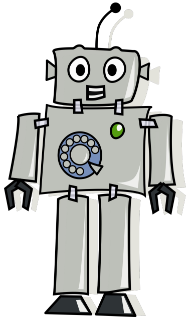
- An **agent** is learning if it **improves its performance on future tasks after making observations** about the world
- Learning can range from trivial to profound:
    - jotting down a phone number (all of us can do this)
    - infer a new theory of the Universe (only a few such as Einstein can do this)

What is Learning?

- An **agent** is learning if it **improves its performance on future tasks after making observations** about the world
- Learning can range from trivial to profound:
  - jotting down a phone number (all of us can do this)
  - infer a new theory of the Universe (only a few such as Einstein can do this)
- We will focus on one class of learning problem
  - “from a collection of input–output pairs, learn a function that predicts the output for new inputs”
- This class of learning task seems restricted but actually has vast applicability
  - Examples?

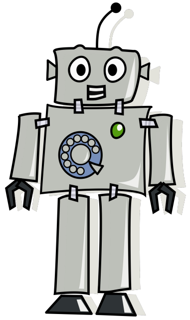
18.1 Forms of Learning

- Any component of an agent can be improved by learning from data
- The improvements, and the improvements techniques, depend on four major factors:
  a. Which component is to be improved
     - workout improves muscles not mind power



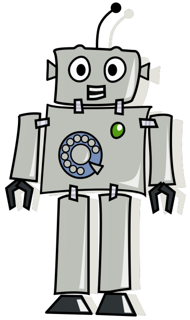
18.1 Forms of Learning

- Any component of an agent can be improved by learning from data
- The improvements, and the improvements techniques, depend on four major factors:
  a. Which component is to be improved
     - workout improves muscles not mind power
  b. What prior knowledge the agent already has
     - is the robot pretrained?



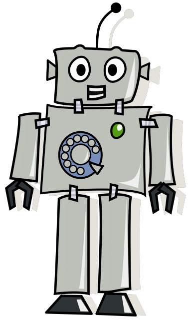
18.1 Forms of Learning

- Any component of an agent can be improved by learning from data
- The improvements, and the improvements techniques, depend on four major factors:
  a. Which component is to be improved
     - workout improves muscles not mind power
  b. What prior knowledge the agent already has
     - is the robot already pretrained?
  c. What representation is used for the data and the component
     - model of human brain?



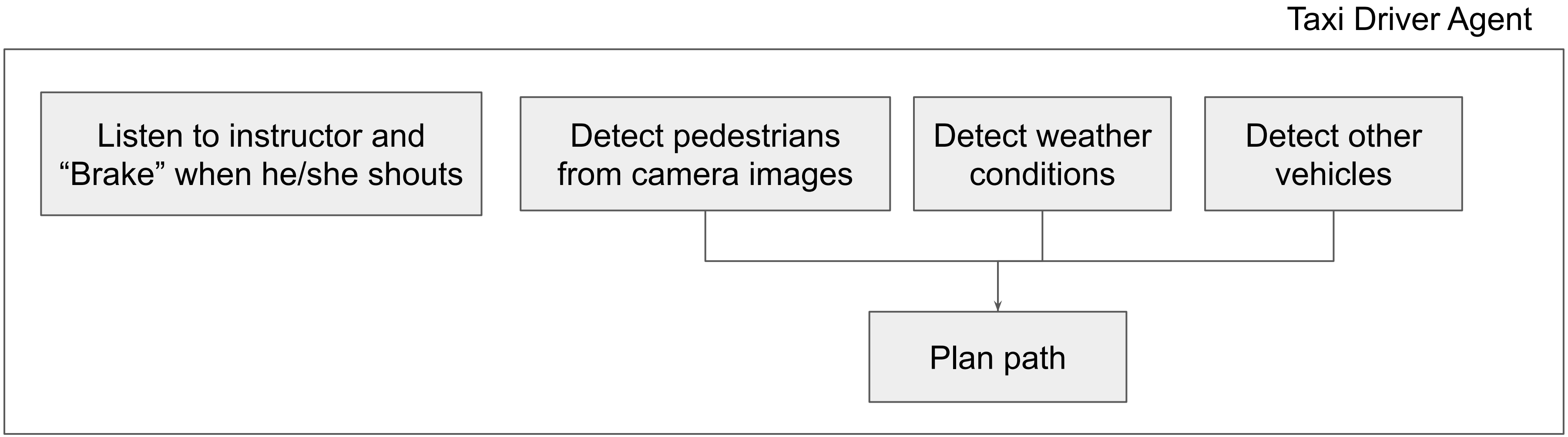
18.1 Forms of Learning

- Any component of an agent can be improved by learning from data
- The improvements, and the improvements techniques, depend on four major factors:
  a. Which component is to be improved
     - workout improves muscles not mind power
  b. What prior knowledge the agent already has
     - is the robot already pretrained?
  c. What representation is used for the data and the component
     - model of human brain?
  d. What feedback is available to learn from
     - reinforcement learning?



18.1 Components to be Learned

- (Although some agents can be end-to-end) An agent may have many components
- Which component do we want to learn at a time?



18.1 Representation

- Examples of representations for agent components:
  - Propositional and first-order logical sentences for the components in a logical agent
  - Bayesian networks for the inferential components of a decision-theoretic agent

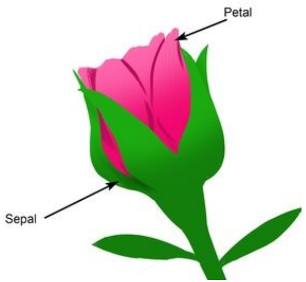
18.1 Representation

- Examples of representations for agent components:
  - Propositional and first-order logical sentences for the components in a logical agent
  - Bayesian networks for the inferential components of a decision-theoretic agent
- Most of current machine learning research covers inputs that form a **factored representation**—a vector of attribute values—and outputs that can be either a continuous numerical value or a discrete value

18.1 Representation

- Examples of representations for agent components:
  - Propositional and first-order logical sentences for the components in a logical agent
  - Bayesian networks for the inferential components of a decision-theoretic agent
- Most of current machine learning research covers inputs that form a **factored representation**—a vector of attribute values—and outputs that can be either a continuous numerical value or a discrete value
- The Iris Flower Dataset (1936)
  - The data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor)
  - Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters
  - https://www.kaggle.com/uciml/iris

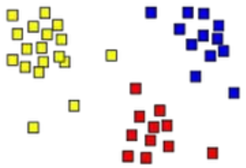
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa
7	4.6	3.4	1.4	0.3	setosa
8	5.0	3.4	1.5	0.2	setosa



18.1 Feedback to Learn From

There are **three** types of feedback that determine the **three main types of learning**:

1. **Unsupervised learning** - the agent learns patterns in the input even though no explicit feedback is supplied

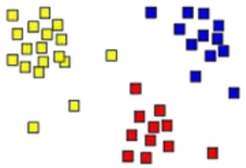


The most common unsupervised learning task is clustering: detecting potentially useful clusters of input examples. For example, a taxi agent might gradually develop a concept of “good traffic days” and “bad traffic days” without ever being given labeled examples of each by a teacher

18.1 Feedback to Learn From

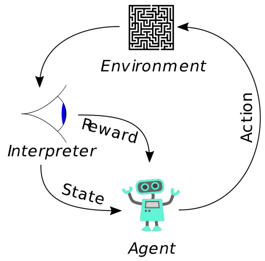
There are **three** types of feedback that determine the three main types of learning:

1. **Unsupervised learning** - the agent learns patterns in the input even though no explicit feedback is supplied



The most common unsupervised learning task is clustering: detecting potentially useful clusters of input examples. For example, a taxi agent might gradually develop a concept of "good traffic days" and "bad traffic days" without ever being given labeled examples of each by a teacher

2. **Reinforcement learning** - the agent learns from a series of reinforcements - rewards or punishments

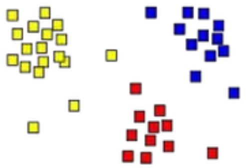


For example, the lack of a tip at the end of the journey gives the taxi agent an indication that it did something wrong. The two points for a win at the end of a chess game tells the agent it did something right. It is up to the agent to decide which of the actions prior to the reinforcement were most responsible for it.

18.1 Feedback to Learn From

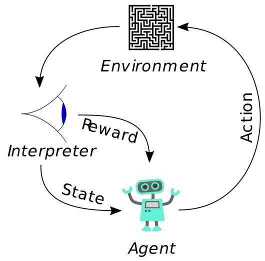
There are **three** types of feedback that determine the three main types of learning:

1. **Unsupervised learning** - the agent learns patterns in the input even though no explicit feedback is supplied



The most common unsupervised learning task is clustering: detecting potentially useful clusters of input examples. For example, a taxi agent might gradually develop a concept of “good traffic days” and “bad traffic days” without ever being given labeled examples of each by a teacher

2. **Reinforcement learning** - the agent learns from a series of reinforcements - rewards or punishments



For example, the lack of a tip at the end of the journey gives the taxi agent an indication that it did something wrong. The two points for a win at the end of a chess game tells the agent it did something right. It is up to the agent to decide which of the actions prior to the reinforcement were most responsible for it.

3. **Supervised learning** - the agent observes some example input–output pairs and learns a function that maps from input to output

The iris dataset example

18.1 Feedback to Learn From

- In practice, the distinctions between supervised, unsupervised, and reinforcement learning are not always so crisp

18.1 Feedback to Learn From

- In practice, the distinctions between supervised, unsupervised, and reinforcement learning are not always so crisp
- In **semi-supervised learning** we are given a few labeled examples and must make what we can of a large collection of unlabeled examples
    - Even the labels themselves may not be the oracular truths that we hope for

18.1 Feedback to Learn From

- In practice, the distinctions between supervised, unsupervised, and reinforcement learning are not always so crisp
- In **semi-supervised learning** we are given a few labeled examples and must make what we can of a large collection of unlabeled examples
    - Even the labels themselves may not be the oracular truths that we hope for
- Example:
    - Imagine that you are trying to build a system to guess a person's age from a photo
    - **Supervised Learning:** You gather some labeled examples by snapping pictures of people and asking their age
    - **Reality:** Some of the people lied about their age
    - It's not just that there is random noise in the data; rather the inaccuracies are systematic, and to uncover them is an unsupervised learning problem involving images, self-reported ages, and true (unknown) ages. Thus, both noise and lack of labels create a continuum between supervised and unsupervised learning.

Unsupervised, Supervised, and Reinforcement Learning

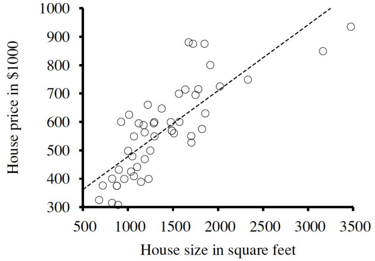
Match the following:

Unsupervised Learning	We have a dataset but there is no target to be predicted. Rather, we want to learn a model that might have generated that set.
Supervised Learning	You teach your kid about different kinds of fruits that are available in world by showing the image of each fruit(X) and its name (Y).
Reinforcement Learning	You ask your child to put apples into different buckets based on size or color.
Unsupervised Learning	This is a setting where we have a sequential decision problem. Making a decision now influences what decisions we can make in the future. A reward function is provided that tells us how "good" certain states are.
Supervised Learning	We have a data set that includes the target values (the values we wish to predict). We try to learn a function that correctly predict the target values from the other features, which can then be used to make predictions about other examples.
Reinforcement Learning	You give apples to your kid in the morning only after brushing the teeth.

https://www.quora.com/What-is-the-difference-between-supervised-unsupervised-reinforcement-and-deep-learning

18.6.1 Univariate Linear Regression

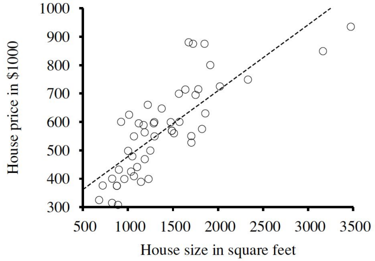
- A univariate linear function (a straight line) with input x and output y has the form
  - $y = w_1 x + w_0$, where w_0 and w_1 are real-valued coefficients to be learned
  - the letter w because we think of the coefficients as weights



18.6.1 Univariate Linear Regression

- A univariate linear function (a straight line) with input x and output y has the form
  - $y = w_1 x + w_0$, where w_0 and w_1 are real-valued coefficients to be learned
  - the letter w because we think of the coefficients as weights
- We define w to be the vector $[w_0,w_1]$, and define

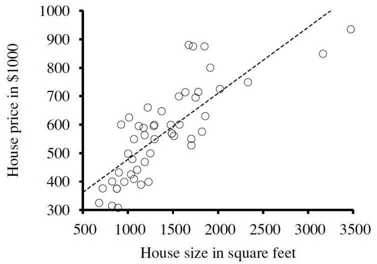
$$h_w(x) = w_1 x + w_0$$



18.6.1 Univariate Linear Regression

- A univariate linear function (a straight line) with input x and output y has the form
  - $y = w_1 x + w_0$, where w_0 and w_1 are real-valued coefficients to be learned
  - the letter w because we think of the coefficients as weights
- We define w to be the vector $[w_0, w_1]$, and define

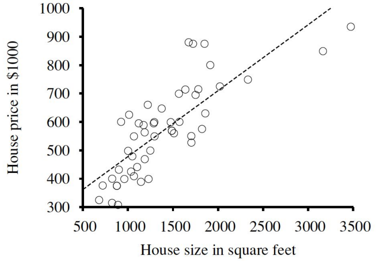
$$h_w(x) = w_1 x + w_0$$



- Example:
  - A training set of n points in the x, y plane, each point representing the size in square feet and the price of a house offered for sale

18.6.1 Univariate Linear Regression

- A univariate linear function (a straight line) with input x and output y has the form
  - $y = w_1 x + w_0$, where w_0 and w_1 are real-valued coefficients to be learned
  - the letter w because we think of the coefficients as weights



- We define w to be the vector $[w_0,w_1]$, and define

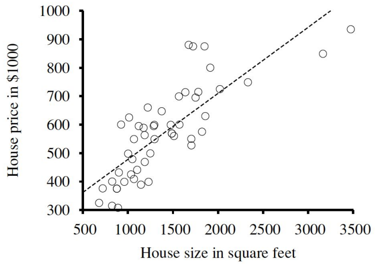
$$h_w(x) = w_1 x + w_0$$

- Example:
  - A training set of n points in the x, y plane, each point representing the size in square feet and the price of a house offered for sale
- The task of finding the h_w that best fits these data is called **linear regression**
  - To fit a line to the data, all we have to do is find the values of the weights $[w_0,w_1]$ that minimize the empirical loss

18.6.1 Univariate Linear Regression

- A univariate linear function (a straight line) with input x and output y has the form
  - $y = w_1 x + w_0$, where w_0 and w_1 are real-valued coefficients to be learned
  - the letter w because we think of the coefficients as weights
- We define w to be the vector $[w_0, w_1]$, and define

$$h_w(x) = w_1 x + w_0$$



- Example:
  - A training set of n points in the x, y plane, each point representing the size in square feet and the price of a house offered for sale
- The task of finding the h_w that best fits these data is called **linear regression**
  - To fit a line to the data, all we have to do is find the values of the weights $[w_0, w_1]$ that minimize the empirical loss
- It is traditional to use the squared loss function, L_2, summed over all the training examples:

$$Loss(h_{\mathbf{w}}) = \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2$$

18.6.1 Univariate Linear Regression

$$Loss(h_{\mathbf{w}}) = \sum_{j=1}^{N}(y_j - (w_1 x_j + w_0))^2$$

- We would like to find $w^* = argmin_w\ Loss\ (h_w)$
- The sum $Loss\ (h_w)$ is minimized when its partial derivatives with respect to w_0 and w_1 are zero

$$\frac{\partial}{\partial w_0}\sum_{j=1}^{N}(y_j - (w_1 x_j + w_0))^2 = 0 \text{ and } \frac{\partial}{\partial w_1}\sum_{j=1}^{N}(y_j - (w_1 x_j + w_0))^2 = 0$$

18.6.1 Univariate Linear Regression

$$Loss(h_{\mathbf{w}}) = \sum_{j=1}^{N}(y_j - (w_1x_j + w_0))^2$$

- We would like to find $w^* = argmin_w\, Loss\,(h_w)$
- The sum $Loss\,(h_w)$ is minimized when its partial derivatives with respect to w_0 and w_1 are zero

$$\frac{\partial}{\partial w_0}\sum_{j=1}^{N}(y_j - (w_1x_j + w_0))^2 = 0 \text{ and } \frac{\partial}{\partial w_1}\sum_{j=1}^{N}(y_j - (w_1x_j + w_0))^2 = 0$$

- These equations have a unique solution:

$$w_1 = \frac{N(\sum x_j y_j) - (\sum x_j)(\sum y_j)}{N(\sum x_j^2) - (\sum x_j)^2}; \quad w_0 = (\sum y_j - w_1(\sum x_j))/N$$

- For example, $w_1 = 0.232$, $w_0 = 246$

18.6.1 Univariate Linear Regression

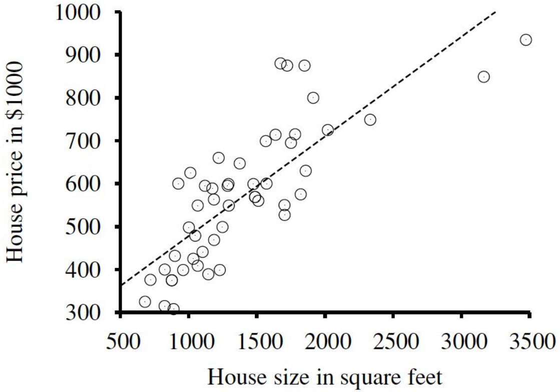
- Many forms of learning involve adjusting weights to minimize a loss
  - so it helps to have a mental picture of what's going on in the **weight space**
- Weight space is defined by all possible settings of the weights

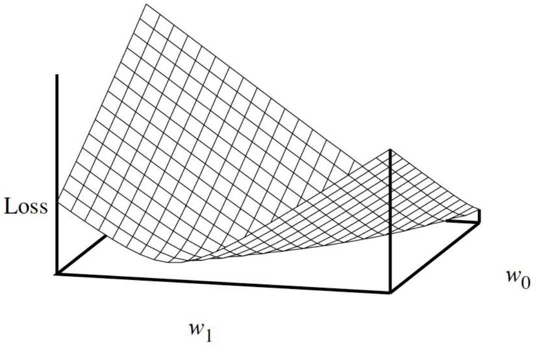
18.6.1 Univariate Linear Regression

- Many forms of learning involve adjusting weights to minimize a loss
  - so it helps to have a mental picture of what's going on in the **weight space**
- Weight space is defined by all possible settings of the weights
- For univariate ('only 1 variable as input') linear regression, the weight space defined by w_0 and w_1 is two-dimensional
  - We can graph the loss as a function of w_0 and w_1 in a 3D plot
  - The loss function is convex

18.6.1 Univariate Linear Regression

- Many forms of learning involve adjusting weights to minimize a loss
  - so it helps to have a mental picture of what's going on in the **weight space**
- Weight space is defined by all possible settings of the weights
- For univariate ('only 1 variable as input') linear regression, the weight space defined by w_0 and w_1 is two-dimensional
  - We can graph the loss as a function of w_0 and w_1 in a 3D plot
  - The loss function is convex





Open in Colab

Request DEMO

Univariate Linear Regression

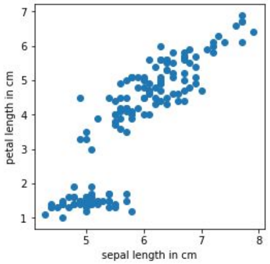
- We would like to predict "sepal width" using "sepal length" on the Iris Flower Dataset
- Dataset: rawdata and metadata

In [0]:
```
from keras.models import Sequential
from keras.layers import Dense
import numpy as np
import matplotlib.pyplot as plt
```

In [16]:
```
datapath = 'https://raw.githubusercontent.com/badriadhikari/2019-Fall-AI/master/MODULE-I/iris.data'

# Column 1. sepal length in cm (load as col 0)
# Column 2. petal length in cm (load as col 1)
# Use loadtxt instead when there are non-numeric values as well
dataset = np.genfromtxt(datapath, delimiter=",", usecols=(0, 2))

print('')
print(dataset.shape)
print('')
print(dataset[0:5])
```

```
(150, 2)

[[5.1 1.4]
 [4.9 1.4]
 [4.7 1.3]
 [4.6 1.5]
 [5.  1.4]]
```

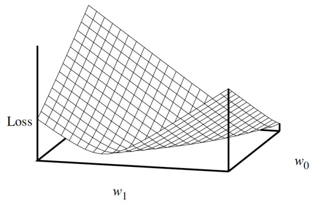
In [17]:
```
plt.figure(figsize=(4,4))
plt.scatter(dataset[:, 0], dataset[:, 1])
plt.xlabel('sepal length in cm')
plt.ylabel('petal length in cm')
plt.show()
```



18.6.1 Beyond Linear Models (for Regression)

$$Loss(h_{\mathbf{w}}) = \sum_{j=1}^{N}(y_j - (w_1 x_j + w_0))^2$$

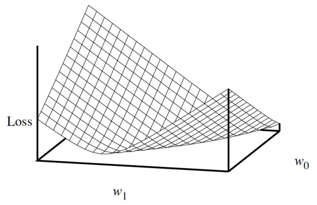
- Equations defining minimum loss will often have no closed-form solution
  - For example, when you decrease w_0 the loss may behave unpredictably



18.6.1 Beyond Linear Models (for Regression) (multivariate)

$$Loss(h_{\mathbf{w}}) = \sum_{j=1}^{N}(y_j - (w_1 x_j + w_0))^2$$

- Equations defining minimum loss will often have no closed-form solution
  - For example, when you decrease w_0 the loss may behave unpredictably
- Here, we face a general optimization search problem in a continuous weight space
  - Such problems can be addressed using algorithms such as Hill-Climbing algorithm that follows the **gradient** of the function to be optimized.
  - Specifically, because we are trying to minimize loss, we can use **gradient descent**



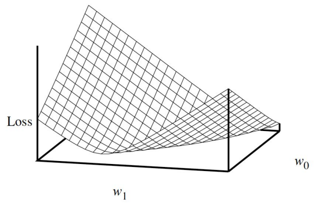
18.6.1 Beyond Linear Models (for Regression) (multivariate)

$$Loss(h_{\mathbf{w}}) = \sum_{j=1}^{N}(y_j - (w_1 x_j + w_0))^2$$

- Equations defining minimum loss will often have no closed-form solution
  - For example, when you decrease w_0 the loss may behave unpredictably
- Here, we face a general optimization search problem in a continuous weight space
  - Such problems can be addressed using algorithms such as Hill-Climbing algorithm that follows the **gradient** of the function to be optimized.
  - Specifically, because we are trying to minimize loss, we can use **gradient descent**

```
w ← any point in the parameter space
loop until convergence do
    for each wi in w do
```

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w})$$



The parameter α, is called the **learning rate** when we are trying to minimize loss in a learning problem. It can be a fixed constant, or it can decay over time as the learning process proceeds.

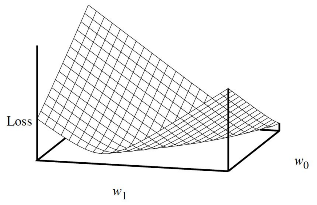
18.6.1 Beyond Linear Models (for Regression) (multivariate)

$$Loss(h_{\mathbf{w}}) = \sum_{j=1}^{N}(y_j - (w_1 x_j + w_0))^2$$

- Equations defining minimum loss will often have no closed-form solution
  - For example, when you decrease w_0 the loss may behave unpredictably
- Here, we face a general optimization search problem in a continuous weight space
  - Such problems can be addressed using algorithms such as Hill-Climbing algorithm that follows the **gradient** of the function to be optimized.
  - Specifically, because we are trying to minimize loss, we can use **gradient descent**

```
w ← any point in the parameter space
loop until convergence do
    for each w_i in w do
```

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w})$$



The parameter α, is called the **learning rate** when we are trying to minimize loss in a learning problem. It can be a fixed constant, or it can decay over time as the learning process proceeds.

For univariate regression, this reduces to: $w_0 \leftarrow w_0 + \alpha \sum_j (y_j - h_{\mathbf{w}}(x_j))$; $w_1 \leftarrow w_1 + \alpha \sum_j (y_j - h_{\mathbf{w}}(x_j)) \times x_j$

Calculating Weights in Univariate Linear Regression

- Solving the equations with derivative of loss function equal to zero, we saw that we can calculate the two weights

$$\frac{\partial}{\partial w_0}\sum_{j=1}^{N}(y_j-(w_1x_j+w_0))^2=0 \text{ and } \frac{\partial}{\partial w_1}\sum_{j=1}^{N}(y_j-(w_1x_j+w_0))^2=0$$

$$w_1=\frac{N(\sum x_jy_j)-(\sum x_j)(\sum y_j)}{N(\sum x_j^2)-(\sum x_j)^2};\quad w_0=(\sum y_j - w_1(\sum x_j))/N$$

- The gradient descent method also gives us an equation to calculate the weights

$$w_i \leftarrow w_i - \alpha\frac{\partial}{\partial w_i}Loss(\mathbf{w})$$

$$w_0 \leftarrow w_0+\alpha\sum_j(y_j-h_{\mathbf{w}}(x_j))\,;\quad w_1\leftarrow w_1+\alpha\sum_j(y_j-h_{\mathbf{w}}(x_j))\times x_j$$

- Both methods provide us a technique to calculate the weights
    - What is the difference between the two methods?
    - Which is more general?

Classroom Discussion & Demo

Request DEMO

Open in Colab

Linear Regression with Two Input Variables using the Iris Flower Dataset

- We would like to predict "petal width" (column 4 in the original dataset) using sepal width (column 2) and petal length (column 3)
- Dataset: rawdata and metadata

In [2]:
```
from keras.models import Sequential
from keras.layers import Dense
import numpy as np
import matplotlib.pyplot as plt

# Column 2. sepal width in cm (load as col 0)
# Column 3. petal length in cm (load as col 1)
# Column 4. petal width in cm (load as col 2)
datapath = 'https://raw.githubusercontent.com/badriadhikari/2019-Fall-AI/master/MODULE-I/iris.dat
a'
dataset = np.genfromtxt(datapath, delimiter=",", usecols=(1, 2, 3))

print('')
print(dataset.shape)
print('')
print(dataset[0:5])
```

```
(150, 3)

[[3.5 1.4 0.2]
 [3.  1.4 0.2]
 [3.2 1.3 0.2]
 [3.1 1.5 0.2]
 [3.6 1.4 0.2]]
```

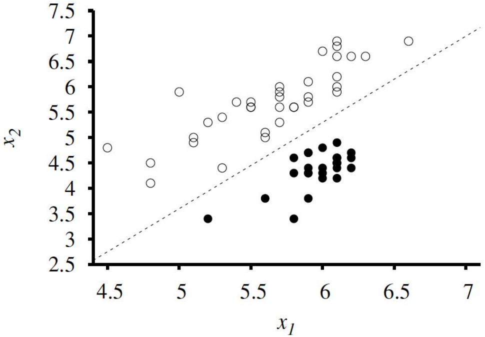
In [5]:
```
# Q1. Why is shuffling important before splitting?
np.random.shuffle(dataset)
print('')
print(dataset[0:5])
train = dataset[:100]
valid = dataset[100:]
print('')
print(train.shape)
print('')
print(valid.shape)
```

```
[[3.7 1.5 0.2]
 [3.  4.2 1.5]
 [2.8 6.7 2. ]
 [2.4 3.7 1. ]
 [3.  1.4 0.2]]

(100, 3)

(50, 3)
```

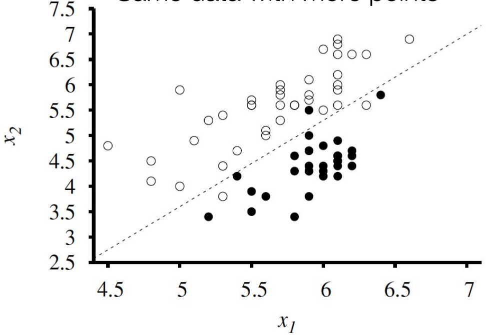
Find a partner!
Answer the questions!

THINK PAIR SHARE

18.6.3 Linear Classifiers with a Hard Threshold

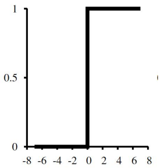


Same data with more points

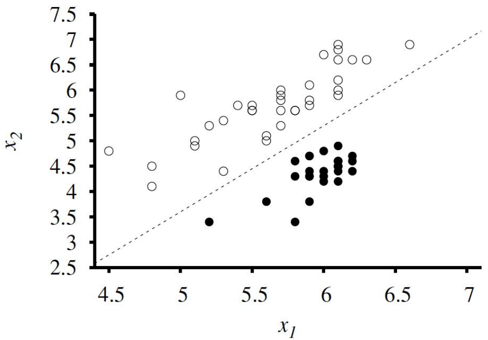


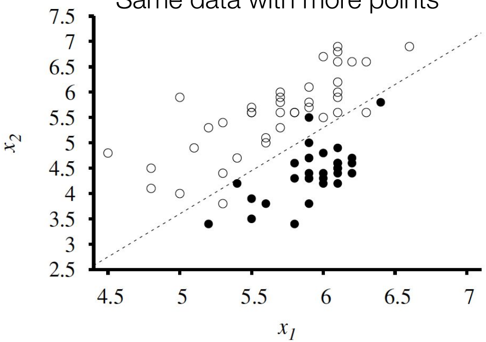
Plot of two seismic data parameters, body wave magnitude x_1 and surface wave magnitude x_2, for earthquakes (white circles) and nuclear explosions (black circles) occurring between 1982 and 1990 in Asia and the Middle East

Linear functions can be used to do classification as well (with the help of a threshold).



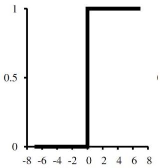
18.6.3 Linear Classifiers with a Hard Threshold





Plot of two seismic data parameters, body wave magnitude x_1 and surface wave magnitude x_2, for earthquakes (white circles) and nuclear explosions (black circles) occurring between 1982 and 1990 in Asia and the Middle East

Linear functions can be used to do classification as well (with the help of a threshold).



A **decision boundary** is a line (or a surface, in higher dimensions) that separates the two classes.

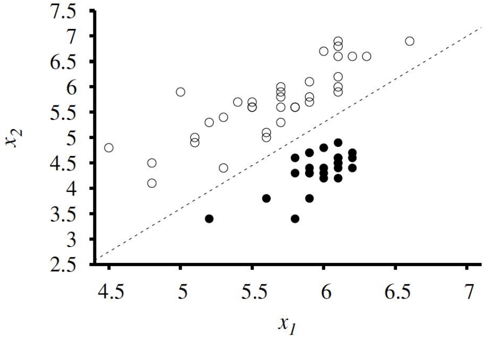
The decision boundary is a straight line in the first figure.

A linear decision boundary is called a **linear separator** and data that admit such a separator are called **linearly separable**.

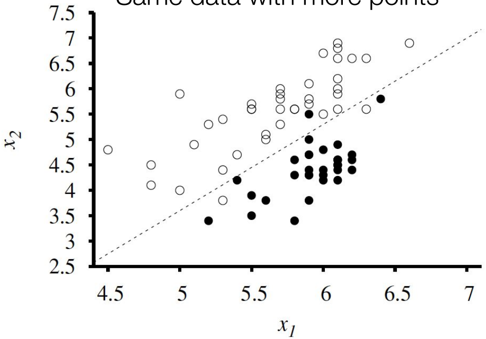
What are the differences between the two pairs of plots?

THINK PAIR SHARE

18.6.3 Linear Classifiers with a Hard Threshold



Same data with more points

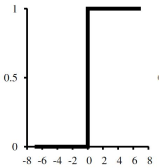


Plot of two seismic data parameters, body wave magnitude x_1 and surface wave magnitude x_2, for earthquakes (white circles) and nuclear explosions (black circles) occurring between 1982 and 1990 in Asia and the Middle East

$x_2 = 1.7\, x_1 - 4.9$ or $-4.9 + 1.7\, x_1 - x_2 = 0$

The explosions, which we want to classify with value 1, are to the right of this line with higher values of x_1 and lower values of x_2, so they are points for which $-4.9 + 1.7x_1 - x_2 > 0$, while earthquakes have $-4.9 + 1.7x_1 - x_2 < 0$.

Linear functions can be used to do classification as well (with the help of a threshold).



A **decision boundary** is a line (or a surface, in higher dimensions) that separates the two classes.

The decision boundary is a straight line in the first figure.

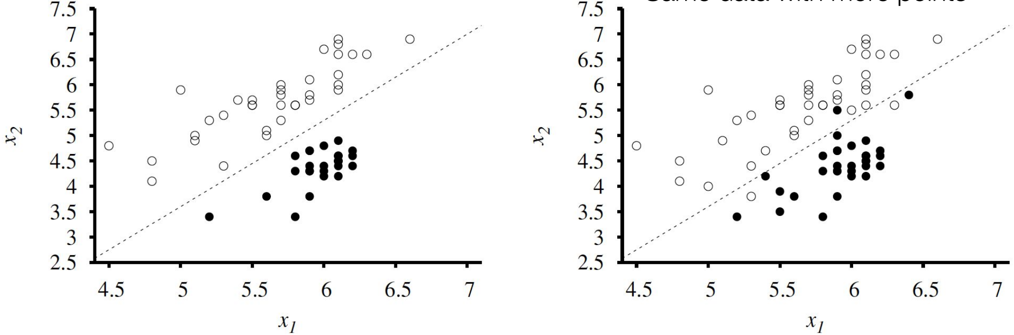
A linear decision boundary is called a **linear separator** and data that admit such a separator are called **linearly separable**.

What are the differences between the two pairs of plots?

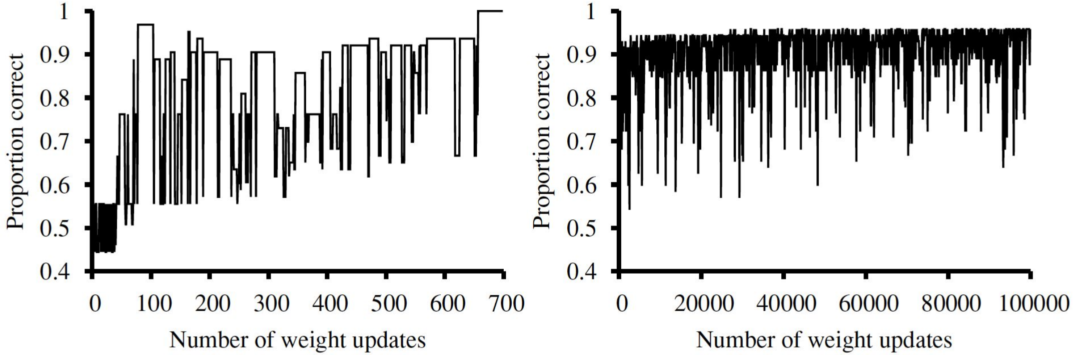
THINK PAIR SHARE

18.6.3 Linear Classifiers with a Hard Threshold

Same data with more points

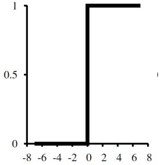


Plot of two seismic data parameters, body wave magnitude x_1 and surface wave magnitude x_2, for earthquakes (white circles) and nuclear explosions (black circles) occurring between 1982 and 1990 in Asia and the Middle East



Plot of total training-set accuracy vs. number of iterations through the training set

Linear functions can be used to do classification as well (with the help of a threshold).



A **decision boundary** is a line (or a surface, in higher dimensions) that separates the two classes.

The decision boundary is a straight line in the first figure.

A linear decision boundary is called a **linear separator** and data that admit such a separator are called **linearly separable**.

What are the differences between the two pairs of plots?

THINK PAIR SHARE

18.6.4 Linear Classification with Logistic Regression

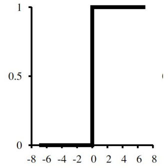
Passing the output of a linear function through the threshold function creates a linear classifier, yet the hard nature of the threshold (e.g. 0) causes some **problems**:

1) The hypothesis $h_w(x)$ is a discontinuous function of its inputs and weights

- $h_w(x)$ = Threshold $(w \cdot x)$, where Threshold $(z) = 1$ if $z \geq 0$ and 0 otherwise

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w})$$

- Discontinuous functions are not differentiable
- This makes learning with the perceptron rule (gradient descent) very unpredictable



18.6.4 Linear Classification with Logistic Regression

Passing the output of a linear function through the threshold function creates a linear classifier, yet the hard nature of the threshold (e.g. 0) causes some **problems**:

1) The hypothesis $h_w(x)$ is a discontinuous function of its inputs and weights

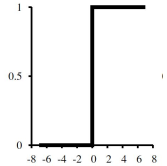
- $h_w(x)$ = Threshold (w · x), where Threshold (z) = 1 if z ≥ 0 and 0 otherwise

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w})$$

- Discontinuous functions are not differentiable
- This makes learning with the perceptron rule (gradient descent) very unpredictable

2) The linear classifier always announces a completely confident prediction of 1 or 0, even for examples that are very close to the boundary

- In many situations, we really need more gradated predictions



18.6.4 Linear Classification with Logistic Regression

Passing the output of a linear function through the threshold function creates a linear classifier, yet the hard nature of the threshold (e.g. 0) causes some **problems**:

1) The hypothesis $h_w(x)$ is a discontinuous function of its inputs and weights

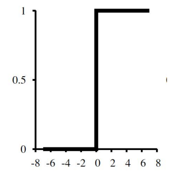
- $h_w(x)$ = Threshold ($w \cdot x$), where Threshold (z) = 1 if $z \geq 0$ and 0 otherwise

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w})$$

- Discontinuous functions are not differentiable
- This makes learning with the perceptron rule (gradient descent) very unpredictable

2) The linear classifier always announces a completely confident prediction of 1 or 0, even for examples that are very close to the boundary

- In many situations, we really need more gradated predictions



These issues can be resolved to a large extent by softening the threshold function—approximating the hard threshold with a continuous, differentiable function

18.6.4 Linear Classification with Logistic Regression

$$Logistic(z) = \frac{1}{1 + e^{-z}}$$

$$h_{\mathbf{w}}(\mathbf{x}) = Logistic(\mathbf{w} \cdot \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}$$

The process of fitting the weights of this model to minimize loss on a data set is **logistic regression.**

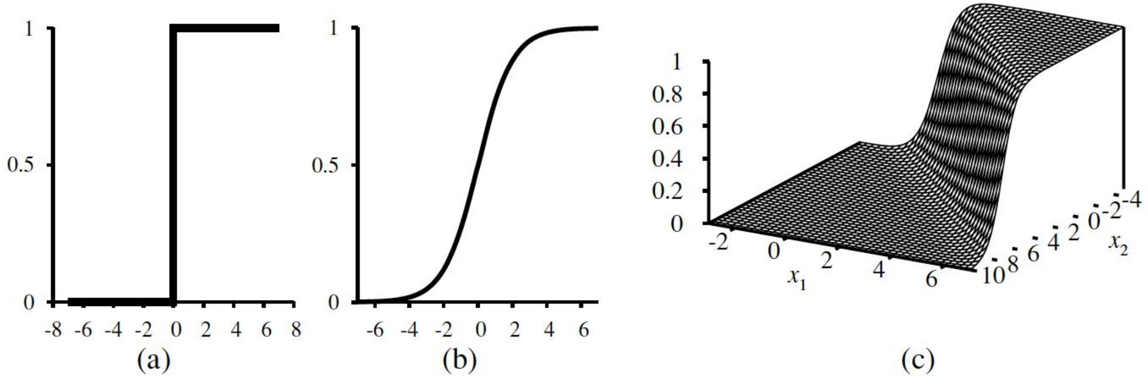


Figure 18.17 (a) The hard threshold function $Threshold(z)$ with 0/1 output. Note that the function is nondifferentiable at $z=0$. (b) The logistic function, $Logistic(z) = \frac{1}{1+e^{-z}}$, also known as the sigmoid function. (c) Plot of a logistic regression hypothesis $h_{\mathbf{w}}(\mathbf{x}) = Logistic(\mathbf{w} \cdot \mathbf{x})$ for the data shown in Figure 18.15(b).

18.6.4 Updating Weights in Logistic Regression (multivariate)

$$h_{\mathbf{w}}(\mathbf{x}) = Logistic(\mathbf{w} \cdot \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w} \cdot \mathbf{x}}}$$

Loss function for Logistic Regression

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w})$$

Perceptron Learning rule (gradient descent)

$$\frac{\partial}{\partial w_i} Loss(\mathbf{w}) = \frac{\partial}{\partial w_i} (y - h_{\mathbf{w}}(\mathbf{x}))^2$$

Derivative of the Loss function

How to calculate this?

Power function rule:

$$\frac{d(g^n)}{dx} = n g^{n-1} \frac{dg}{dx}$$

Chain rule:

$$\partial g(f(x))/\partial x = g'(f(x))\, \partial f(x)/\partial x.)$$

For logistic function:

L'(x) = L(x) (1 - L(x))

18.6.4 Updating Weights in Logistic Regression (multivariate)

$$\frac{\delta}{\delta_{w_i}} Loss(w) = \frac{\delta}{\delta_{w_i}} (y - h_w(x))^2$$

$$= 2(y - h_w(x)) \frac{\delta}{\delta_{w_i}} (y - h_w(x))$$

from power function rule

$$\frac{\delta}{\delta_x} g(x)^n = n[g(x)]^{n-1} \frac{\delta}{\delta_x}(g(x))$$

$$= 2(y - h_w(x)) \frac{\delta}{\delta_{w_i}} (-h_w(x))$$

$$= -2(y - h_w(x)) \frac{\delta}{\delta_{w_i}} (h_w(x))$$

$$h_w(x) = \frac{1}{1+e^{-w.x}} = g(f(x))$$

$$\frac{1}{1+e^{-x}} \qquad w.x$$

$$= -2(y - h_w(x)) \frac{\delta}{\delta_{w.x}} g(w.x) \cdot \frac{\delta}{\delta_{w_i}} (w.x)$$

from chain rule

$$\frac{\delta}{\delta_x} g(f(x)) = g'f(x) \cdot \frac{\delta}{\delta_x} f(x)$$

$$= -2(y - h_w(x)) [g'(w.x)] \cdot \frac{\delta}{\delta_{w_i}} (w_1 x_1 + w_2 x_2 + w_i x_i + \ldots + w_n x_n)$$

$$= -2(y - h_w(x)) [g(w.x)(1 - g(w.x))] \, x_i$$

derivative of a logistic function

$$\frac{\delta}{\delta_x}\left(\frac{1}{1+e^{-x}}\right) = \frac{1}{1+e^{-x}}\left(1 - \frac{1}{1+e^{-x}}\right)$$

$$\therefore \frac{\delta}{\delta_{w_i}} Loss(w) = -2(y - h_w(x)) \, h_w(x) \, (1 - h_w(x)) \, x_i$$

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w})$$

$$w_i \leftarrow w_i + \alpha \, (y - h_{\mathbf{w}}(\mathbf{x})) \times h_{\mathbf{w}}(\mathbf{x})(1 - h_{\mathbf{w}}(\mathbf{x})) \times x_i$$

18.6.4 Linear Classification with Logistic Regression

Additional **advantages of Logistic Regression** (compared to hard threshold):

1. In a linearly separable case, logistic regression is somewhat slower to converge, but behaves much more predictably

18.6.4 Linear Classification with Logistic Regression

Additional **advantages of Logistic Regression** (compared to hard threshold):

1. In a linearly separable case, logistic regression is somewhat slower to converge, but behaves much more predictably
2. Where the data are noisy and nonseparable, logistic regression converges far more quickly and reliably; these advantages tend to carry over into real-world applications

18.6.4 Linear Classification with Logistic Regression

Additional **advantages of Logistic Regression** (compared to hard threshold):

1. In a linearly separable case, logistic regression is somewhat slower to converge, but behaves much more predictably
2. Where the data are noisy and nonseparable, logistic regression converges far more quickly and reliably; these advantages tend to carry over into real-world applications

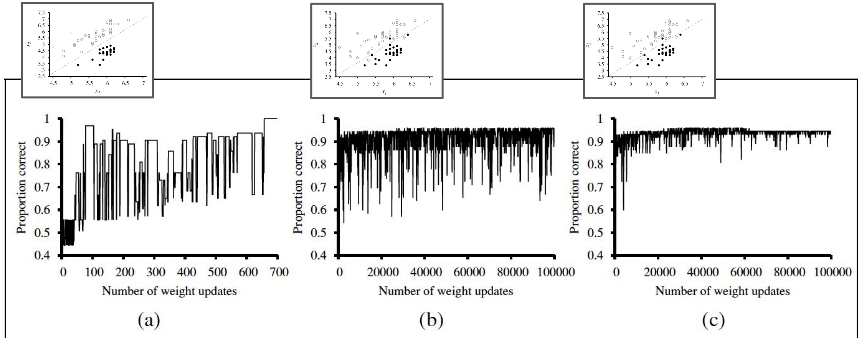


Figure 18.16 (a) Plot of total training-set accuracy vs. number of iterations through the training set for the perceptron learning rule, given the earthquake/explosion data in Figure 18.15(a). (b) The same plot for the noisy, non-separable data in Figure 18.15(b); note the change in scale of the x-axis. (c) The same plot as in (b), with a learning rate schedule $\alpha(t) = 1000/(1000 + t)$.

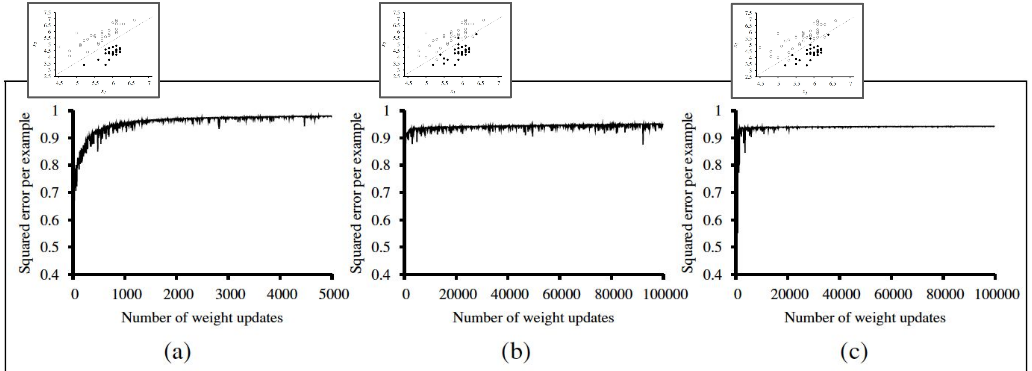


Figure 18.18 Repeat of the experiments in Figure 18.16 using logistic regression and squared error. The plot in (a) covers 5000 iterations rather than 1000, while (b) and (c) use the same scale.

Open in Colab

Logistic Regression Example using the Iris Flower Dataset

- We would like to predict if a given data point (one row) belongs to the 'Iris-setosa' class
- Dataset: rawdata and metadata

In [25]:
```
from keras.models import Sequential
from keras.layers import Dense
import numpy as np
import matplotlib.pyplot as plt

datapath = 'https://raw.githubusercontent.com/badriadhikari/2019-Fall-AI/master/MODULE-I/iris.dat
a'
dataset = np.genfromtxt(datapath, delimiter=",", dtype = str)

print('')
print(dataset.shape)
print('')
print(dataset[0:5])

dataset[:, 4] = np.where(dataset[:, 4] == 'Iris-setosa', 1, 0)
#dataset[:, 4] = np.where(dataset[:, 4] == 'Iris-versicolor', 1, 0)
#dataset[:, 4] = np.where(dataset[:, 4] == 'Iris-virginica', 1, 0)

dataset = dataset.astype(float)
print('')
print(dataset[0:5])

np.random.shuffle(dataset)
print('')
print(dataset[0:5])

train = dataset[:100]
valid = dataset[100:]
print('')
print(train.shape)
print('')
print(valid.shape)
```

```
(150, 5)

[['5.1' '3.5' '1.4' '0.2' 'Iris-setosa']
 ['4.9' '3.0' '1.4' '0.2' 'Iris-setosa']
 ['4.7' '3.2' '1.3' '0.2' 'Iris-setosa']
 ['4.6' '3.1' '1.5' '0.2' 'Iris-setosa']
 ['5.0' '3.6' '1.4' '0.2' 'Iris-setosa']]

[[5.1 3.5 1.4 0.2 1. ]
 [4.9 3.  1.4 0.2 1. ]
 [4.7 3.2 1.3 0.2 1. ]
 [4.6 3.1 1.5 0.2 1. ]
 [5.  3.6 1.4 0.2 1. ]]

[[6.1 2.8 4.  1.3 0. ]
 [5.  3.3 1.4 0.2 1. ]
 [6.4 2.7 5.3 1.9 0. ]
 [5.7 2.5 5.  2.  0. ]
 [4.8 3.4 1.9 0.2 1. ]]

(100, 5)

(50, 5)
```

18.7 Artificial Neural Networks (Motivation)

In 1953, Professor Theodor Erismann devised an experiment

- performing it upon his assistant and student, Ivo Kohler

He made Kohler wear a pair of hand-engineered goggles

- Specially arranged mirrors flipped the light that would reach eyes, top becoming bottom, and bottom top.

After 10 days, Kohler had grown accustomed to the invariably upside-down world

- everything seemed to him normal, rightside-up
- He could do everyday activities in public perfectly well: walk along a crowded sidewalk, even ride a bicycle

https://www.theguardian.com/education/2012/nov/12/improbable-research-seeing-upside-down

18.7 Artificial Neural Networks (Motivation)

Science Contents News Careers Journals

SHARE

REPORTS

Eye-specific termination bands in tecta of three-eyed frogs

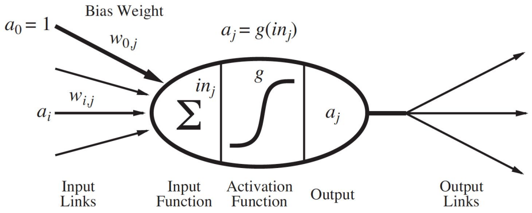
M Constantine-Paton, MI Law
+ See all authors and affiliations

Science 10 Nov 1978:
Vol. 202, Issue 4368, pp. 639-641
DOI: 10.1126/science.309179

Dr. Martha Constantine-Paton is a neuroscientist at MIT

18.7.1 Neural Network Structures

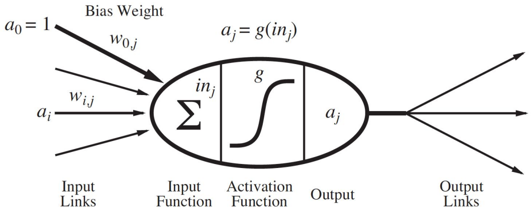
- Neural networks are composed of nodes or units connected by directed links
- A link from unit **i** to unit **j** serves to propagate the activation $\mathbf{a_i}$ from **i** to **j**
- Each link also has a numeric weight $\mathbf{w_{i,j}}$ associated with it, which determines the strength and sign of the connection
- Just as in linear regression models, each unit has a dummy input $a_0 = 1$ with an associated weight $\mathbf{w_{0,j}}$



18.7.1 Neural Network Structures

- Neural networks are composed of nodes or units connected by directed links
- A link from unit **i** to unit **j** serves to propagate the activation $\mathbf{a_i}$ from **i** to **j**
- Each link also has a numeric weight $\mathbf{w_{i,j}}$ associated with it, which determines the strength and sign of the connection
- Just as in linear regression models, each unit has a dummy input $a_0 = 1$ with an associated weight $\mathbf{w_{0,j}}$
- Each unit j first computes a weighted sum of its inputs:

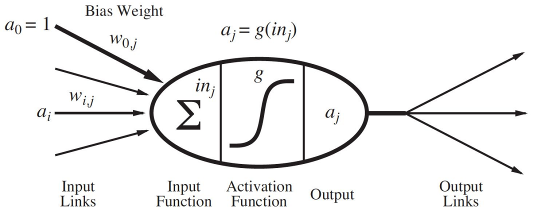
$$in_j = \sum_{i=0}^{n} w_{i,j} a_i$$



18.7.1 Neural Network Structures

- Neural networks are composed of nodes or units connected by directed links
- A link from unit **i** to unit **j** serves to propagate the activation $\mathbf{a_i}$ from **i** to **j**
- Each link also has a numeric weight $\mathbf{w_{i,j}}$ associated with it, which determines the strength and sign of the connection
- Just as in linear regression models, each unit has a dummy input $a_0 = 1$ with an associated weight $\mathbf{w_{0,j}}$
- Each unit j first computes a weighted sum of its inputs:

$$in_j = \sum_{i=0}^{n} w_{i,j} a_i$$



- Then, we apply an **activation function g** to this sum to derive the output:

$$a_j = g(in_j) = g\left(\sum_{i=0}^{n} w_{i,j} a_i\right)$$

18.7.1 Neural Network Structures

There are distinct ways to connect neurons:

1. A feed-forward network has connections only in one direction—that is, it forms a directed acyclic graph
   a. Every node receives input from “upstream” nodes and delivers output to “downstream” nodes; there are no loops
   b. A feed-forward network represents a function of its current input; thus, it has no internal state other than the weights themselves

18.7.1 Neural Network Structures

There are distinct ways to connect neurons:

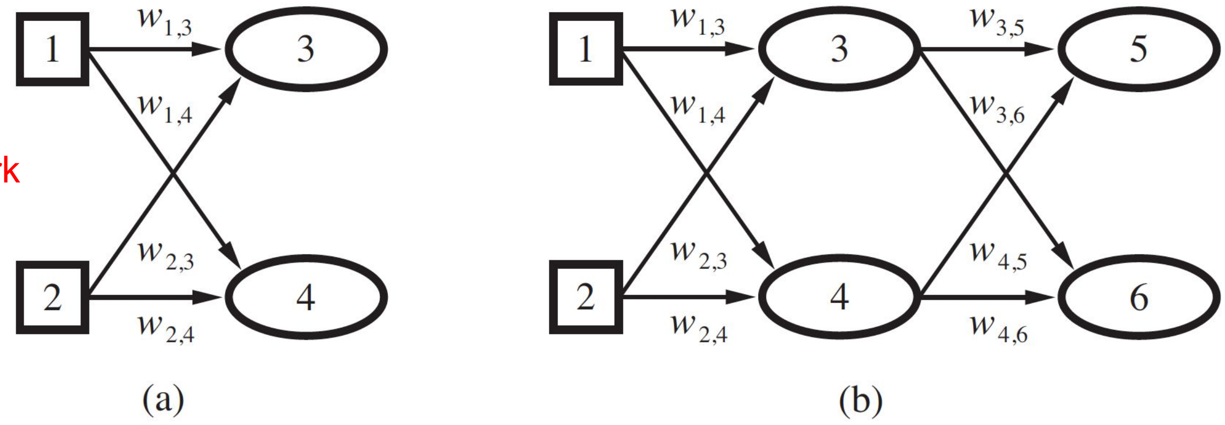
1. A feed-forward network has connections only in one direction—that is, it forms a directed acyclic graph
   a. Every node receives input from "upstream" nodes and delivers output to "downstream" nodes; there are no loops
   b. A feed-forward network represents a function of its current input; thus, it has no internal state other than the weights themselves
2. A recurrent network, on the other hand, feeds its outputs back into its own inputs
   a. This means that the activation levels of the network form a dynamical system that may reach a stable state or exhibit oscillations or even chaotic behavior
   b. Moreover, the response of the network to a given input depends on its initial state, which may depend on previous inputs
   c. Hence, recurrent networks (unlike feed-forward networks) can support short-term memory
   d. This makes them more interesting as models of the brain, but also more difficult to understand

We will focus on feed-forward networks!

18.7.2 Single-layer feed-forward neural networks (Perceptrons)

A network with all the inputs connected directly to the outputs is called a single-layer neural network, or a perceptron network

Perceptron Network



Neural Network

Figure 18.20 (a) A perceptron network with two inputs and two output units. (b) A neural network with two inputs, one hidden layer of two units, and one output unit. Not shown are the dummy inputs and their associated weights.

18.7.2 Single-layer feed-forward neural networks (Perceptrons)

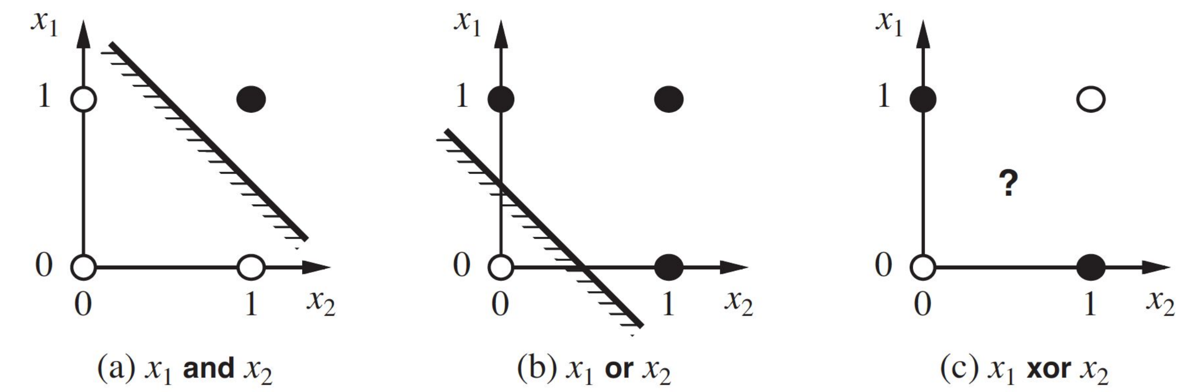
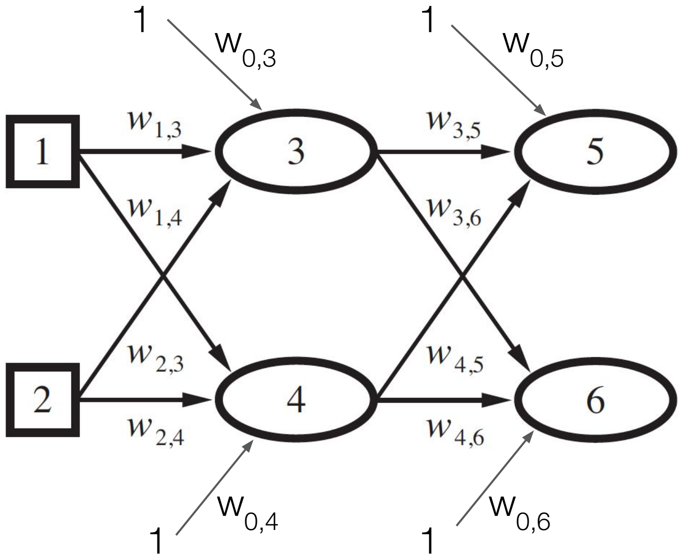


Figure 18.21 Linear separability in threshold perceptrons. Black dots indicate a point in the input space where the value of the function is 1, and white dots indicate a point where the value is 0. The perceptron returns 1 on the region on the non-shaded side of the line. In (c), no such line exists that correctly classifies the inputs.

What is the key limitation of a perceptron network? How to overcome the limitation?

THINK PAIR SHARE

18.7.3 Multilayer Feed-forward Neural Networks



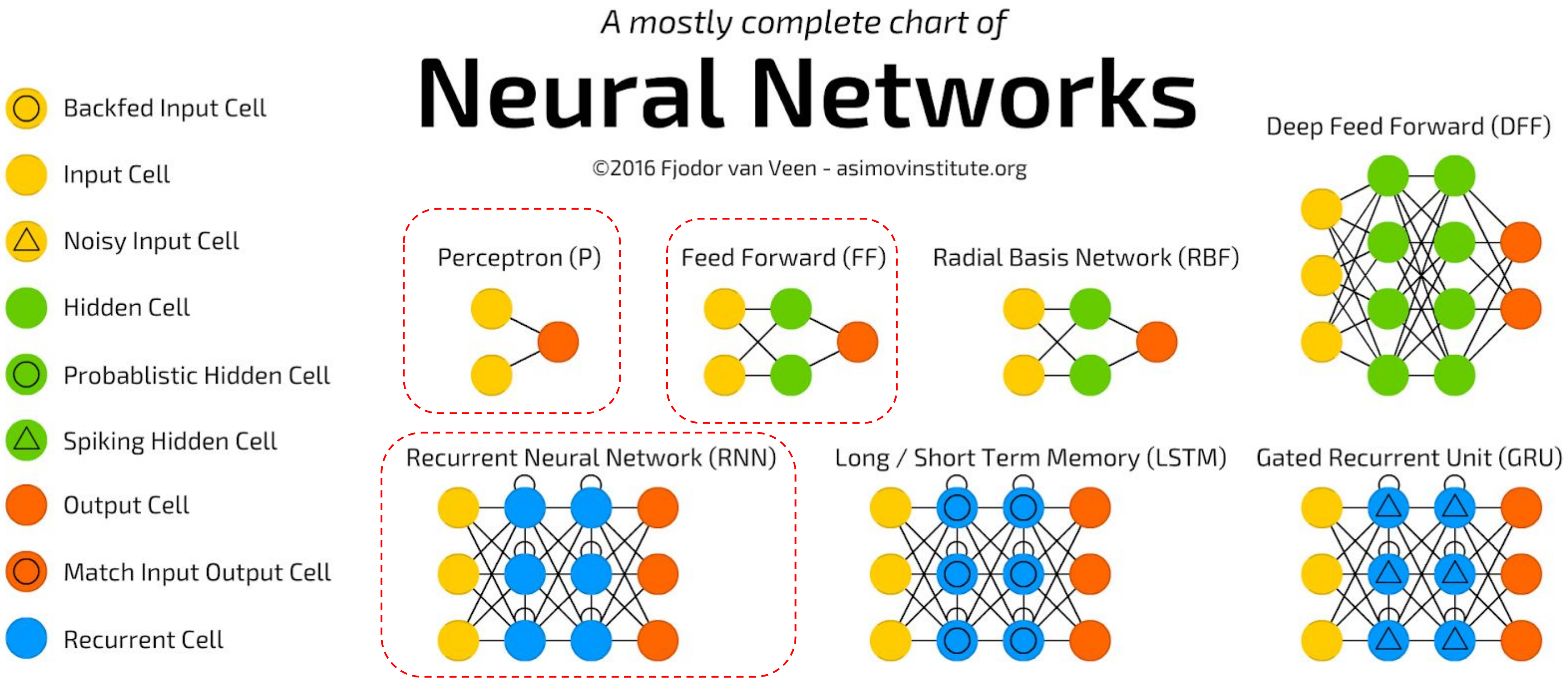
What do a_3 and a_4 stand for?

THINK PAIR SHARE

What are $w_{0,5}$, $w_{0,3}$, $w_{0,4}$, and $w_{0,6}$?

$$
\begin{aligned}
a_5 &= g(w_{0,5,}+w_{3,5}\,a_3 + w_{4,5}\,a_4) \\
&= g(w_{0,5,}+w_{3,5}\,g(w_{0,3} + w_{1,3}\,a_1 + w_{2,3}\,a_2) + w_{4,5}\,g(w_04 + w_{1,4}\,a_1 + w_{2,4}\,a_2)) \\
&= g(w_{0,5,}+w_{3,5}\,g(w_{0,3} + w_{1,3}\,x_1 + w_{2,3}\,x_2) + w_{4,5}\,g(w_04 + w_{1,4}\,x_1 + w_{2,4}\,x_2))
\end{aligned}
$$

Neural Network Structures



18.10 Ensemble Learning

- The idea of ensemble learning methods is to select a collection, or ensemble, of hypotheses from the hypothesis space and combine their predictions; i.e. use multiple models; e.g. boosting

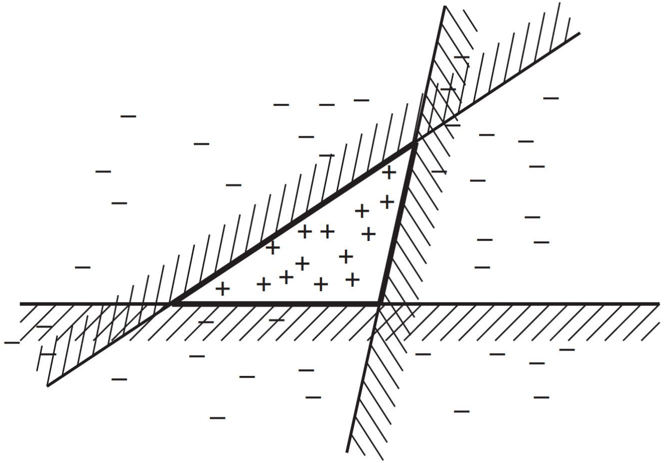


Figure 18.32 Illustration of the increased expressive power obtained by ensemble learning. We take three linear threshold hypotheses, each of which classifies positively on the unshaded side, and classify as positive any example classified positively by all three. The resulting triangular region is a hypothesis not expressible in the original hypothesis space.

Summary

- If the available feedback provides the correct answer for example inputs, then the learning problem is called **supervised learning**. The task is to learn a function $y = h(x)$.
- Learning a discrete-valued function is called **classification**; learning a continuous function is called **regression**.
- Sometimes not all errors are equal. A **loss function** tells us how bad each error is.
- **Logistic regression** replaces the perceptron's hard threshold with a soft threshold defined by a logistic function. Gradient descent works well even for noisy data that are not linearly separable.
- Neural networks represent complex nonlinear functions with a network of linear threshold units.
- Ensemble methods such as **boosting** often perform better than individual methods.