Introduction to Data Science

ESC 403

Lecture 8

Prof. Dr. Robert Feldmann
Department of Astrophysics
robert.feldmann@uzh.ch

Today

- Time series analysis
  - Univariate time series
  - Box-Jenkins / ARIMA
  - Model selection, estimation, validation

- Artificial Neural Networks
  - MCP neuron, Perceptron
  - MLP, universal approximators

Additional material for today's lecture

- Applied Time Series Analysis, Penn State University, https://online.stat.psu.edu/stat510
- Forecasting: Principles & Practice by Rob Hyndman, Geroge Athanasopoulos, https://otexts.com/fpp2/

- http://neuralnetworksanddeeplearning.com
- http://deeplearning.stanford.edu/tutorial
- http://deeplearning.net/tutorial
- Christopher M. Bishop. Neuronal Networks for Pattern Recognition, Springer

Logistics

Course website: OLAT

lms.uzh.ch olat.uzh.ch

- Wiki page
- Forum
- Lecture slides, Exercise sheets
- Dropbox to upload exercises & project proposals

Group Project

- 3 or 4 group members (see, e.g., OLAT Forum)
- write a proposal (see OLAT Wiki for guidance)
- Proposal deadline March 28 ✔
- Project deadline **May 17**

Exercises

- handed out today, return Tuesday next week
- need 50% of points to take exam

Exam

- written exam, pen & paper style
- June 7 (Fri), 10 am — 11.30 am, room: Y15-G-40

Time series: realization of (discrete-time) stochastic process $\{X_t\}_{t=1\ldots n}$

X_1	X_2	X_3	X_4	X_5	X_6
3	1.7	0	-1	2	5

time →

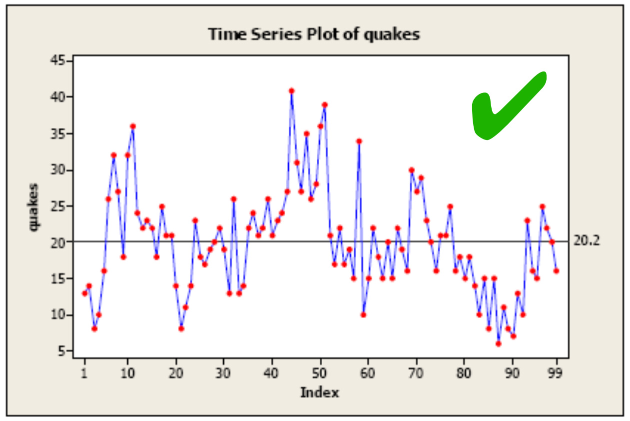
Each X_t is a random variable!

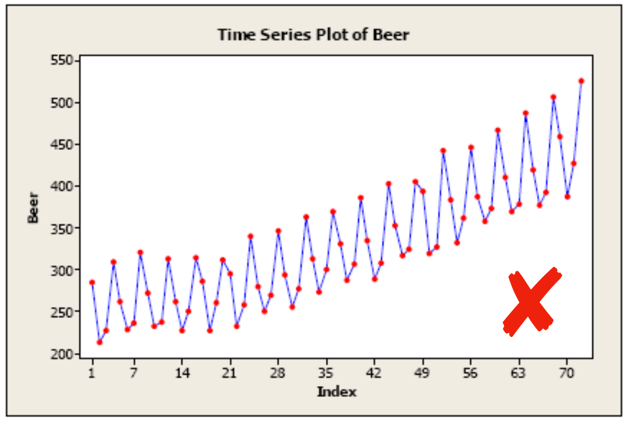
Stationarity

- Mean $\mathrm{E}[X_t]$ is the same for all times t
- Variance $\mathrm{Var}[X_t]$ is the same for all times t
- Covariance between X_t and X_{t-n} is the same for all t (but may vary with lag n)

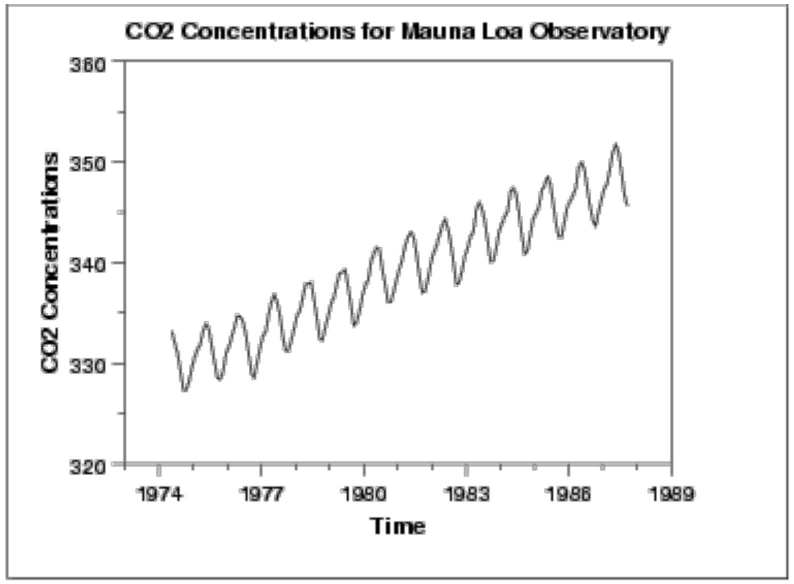
➜ Time series should have

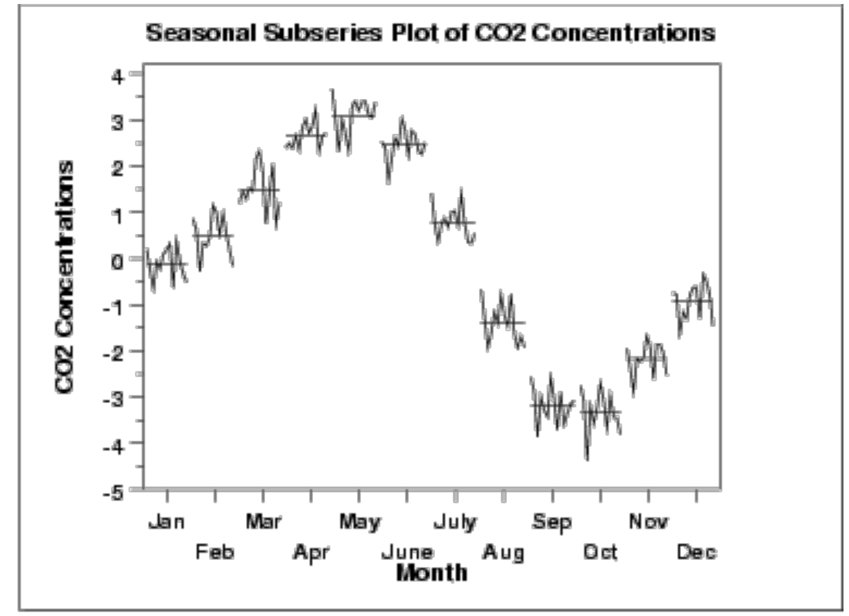
- no obvious trends
- constant variance with time
- constant autocorrelation structure over time
- no periodic fluctuations (no seasonality)



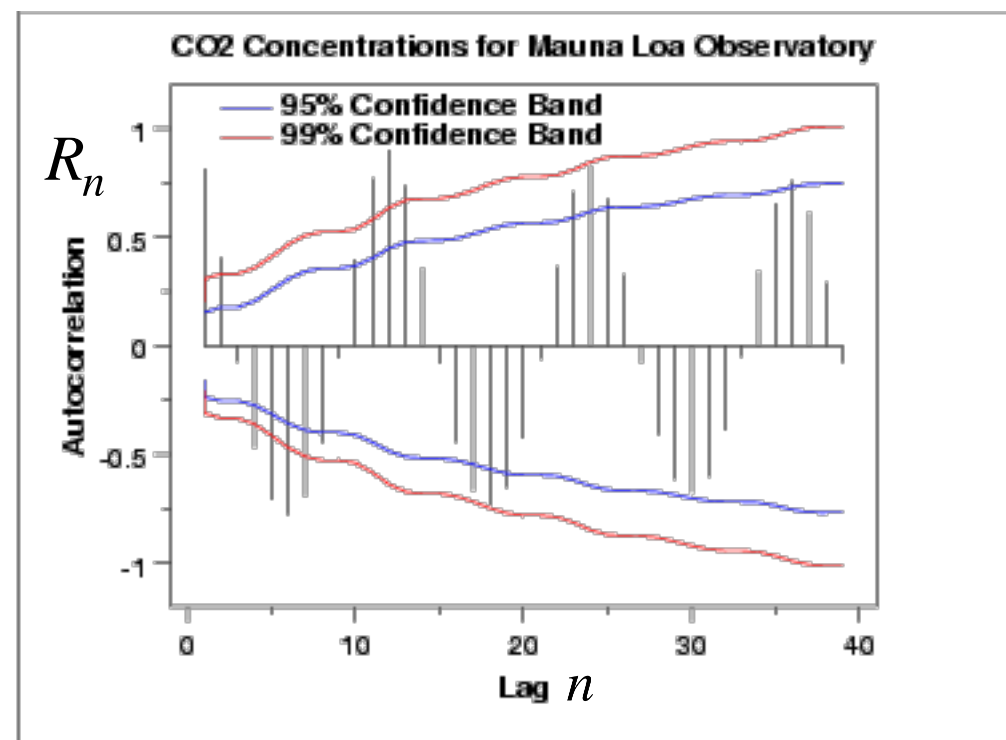


- Seasonal subseries plot after de-trending





- Autocorrelation plot



Autocorrelation

$$\rho(n) = \frac{\mathrm{E}\left[(X_t - \mu)(X_{t+n} - \mu)\right]}{\mathrm{Var}[X_t]} = \frac{\mathrm{Cov}[X_t, X_{t+n}]}{\mathrm{Var}[X_t]}$$

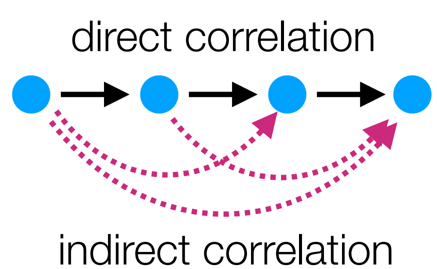
$$R_n = \frac{\sum_{t=1}^{N-n} (x_t - \bar{x})(x_{t+n} - \bar{x})}{\sum_{t=1}^{N} (x_t - \bar{x})^2}$$ with $\bar{x} = \frac{1}{N}\sum_{t=1}^{N} x_t$

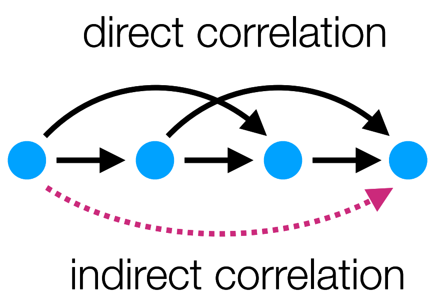
sample autocorrelation

Correlation functions

Autocorrelation function (ACF) $\rho(n) = \dfrac{\mathrm{Cov}[X_t, X_{t+n}]}{\mathrm{Var}[X_t]}$

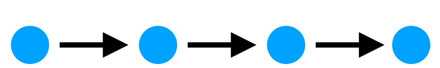
- Autocorrelation at lag n is a measure of how much X_t and X_{t+n} are correlated (for all and any t), but this correlation can be indirect
  - e.g., if X_t and X_{t+1} are correlated (and thus X_{t+1} and X_{t+2}), then usually at some level also X_t and X_{t+2} etc

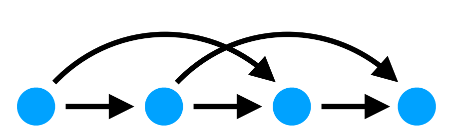




Partial autocorrelation function (PACF) $\rho'(n) = \dfrac{\mathrm{Cov}[X_t, X_{t+n} \,|\, X_{t+1}, \ldots, X_{t+n-1}]}{\mathrm{Var}[X_t \,|\, X_{t+1}, \ldots, X_{t+n-1}]}$

- Partial auto-correlation at lag t is the remaining correlation between X_t and X_{t+n} after all autocorrelations of lag $1, \ldots, n-1$ have been removed
  - e.g., if X_t and X_{t+1} are directly correlated $(\forall t)$, but no other correlations, then the partial autocorrelation between X_t and X_{t+2} is zero





What to do if time series is non-stationary?

If trend:

- Remove trend via a regression, e.g., $Z_t = X_t - \beta \times t$
- Remove trend by differencing the data, new series $Z_t = X_t - X_{t-1}$ for all $t > 1$

If non-constant variance:

- Consider transforming the data, e.g., $Z_t = \ln(X_t)$ or $Z_t = \sqrt{X_t}$ for all t

If periodicity / seasonality:

- *Seasonal differencing*, e.g., if monthly data and 12-month period, then

$$Z_t = X_t - X_{t-12} \qquad \text{for all } t > 12$$

It is okay to run the analysis and build the model with these modified time series. However, at the end one needs to transform back from Z_t to X_t.

Many tools do all of the above for us, often only needed to identify “right" model.

A basic modeling approach

Decomposition: *Time series consists of trend, seasonality and random variations*

- Additive: X_t = Trend + Seasonal + Random
- Multiplicative: X_t = Trend * Seasonal * Random

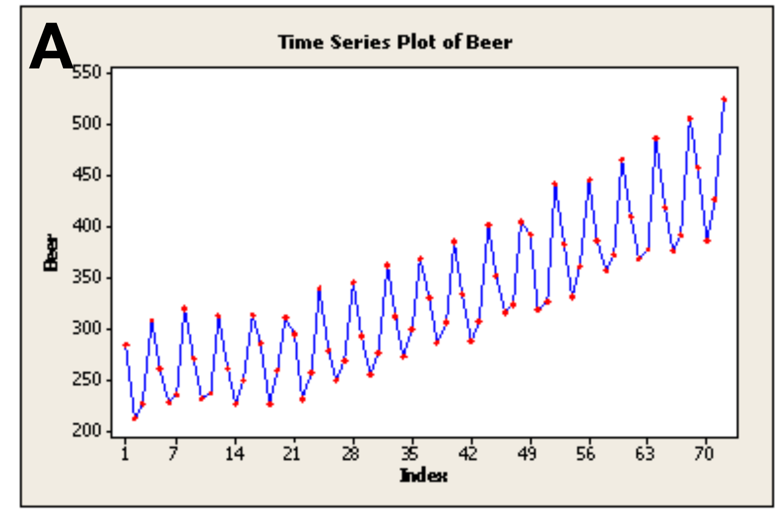
Model Identification:

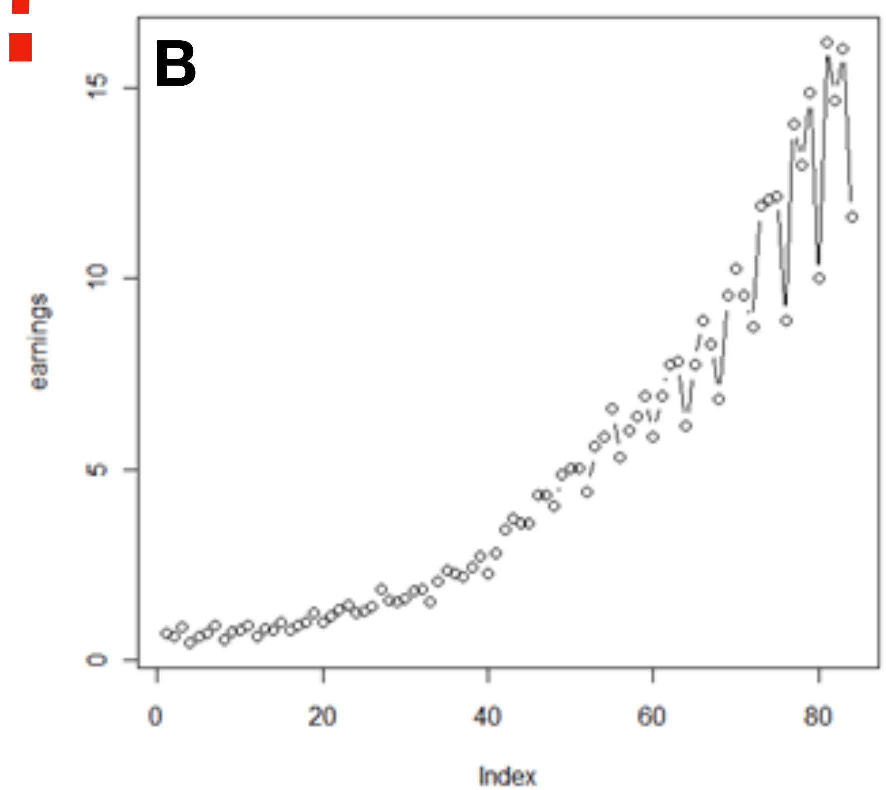
- trend with superimposed seasonality & noise

Model Estimation:

1. Estimate trend (e.g., via smoothing or regression)
2. De-trend series (subtracting for additive, divide for multiplicative)
3. Estimate seasonal factors (e.g., mean value for each month)
4. Remove seasonal factor to get random component (subtract for additive, divide for multiplicative)
5. Potentially repeat steps 1-4 a few times

?





Model Validation:

- 4-plot & autocorrelation of residuals (value of time series - model prediction)

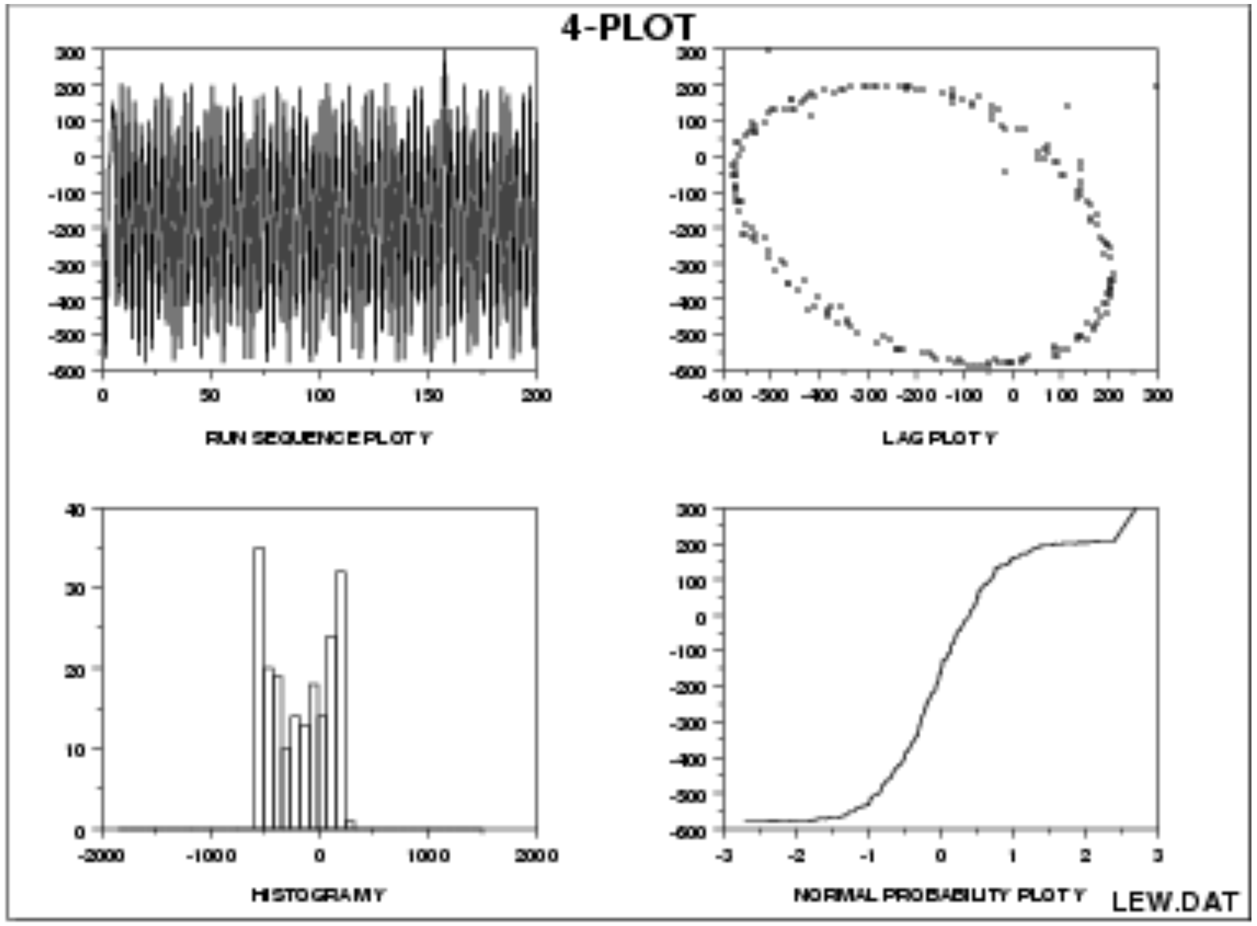
Plots for Model Validation

- Residuals should be white noise drawings (uncorrelated) from a fixed distribution (usually normal) with fixed mean and variance
- If residuals do not satisfy these conditions ➜ model does not work well

<u>4-plot of **residuals**</u>

?

- residuals vs time/index
  - no obvious patterns?
  - no trend & constant variance?
- residuals at t vs residuals at $t-1$
  - no correlation?
- histogram of residuals
  - normal distribution with zero mean?
- Quantile-Quantile plot for normal distribution
  - deviation from straight line?



<u>Autocorrelation plot of **residuals**</u>

- *residuals are uncorrelated?*

<u>Box-Ljung statistical test</u> (acorr_ljungbox in statsmodels Python, Box.test in stats in R)

AR I MA

Autoregressive integrated moving average models

- general class of models for univariate time series (aka Box-Jenkins models)
- flexible, allows to capture a large variety of correlation structures
- Extension: seasonal ARIMA has additional parameters to deal with seasonality

Model

$$X_t = \delta + \phi_1 X_{t-1} + \phi_2 X_{t-2} + \ldots + \phi_p X_{t-p}$$
$$+A_t + \theta_1 A_{t-1} + \theta_2 A_{t-2} + \ldots + \theta_q A_{t-q}$$

$$\delta = \mu - \mu \sum_{i=1}^{p} \phi_i$$
$$\forall t : \mathrm{E}[X_t] = \mu, \mathrm{E}[A_t] = 0$$

X_t time series

A_t white noise from a fixed distribution (often standard normal), zero mean

$\{\phi_j\}_{j=1\ldots p}$	parameters determining how present value depends on past values
$\{\theta_i\}_{i=1\ldots q}$	parameters determining how present value depends on past noise

model parameters to be determined

- Equation above assumes that the time series X_t is stationary
- if X_t not stationary, make it so by differencing (d times) => Integrated models

p, d, q hyper-parameters => complexity of the model

AR I MA

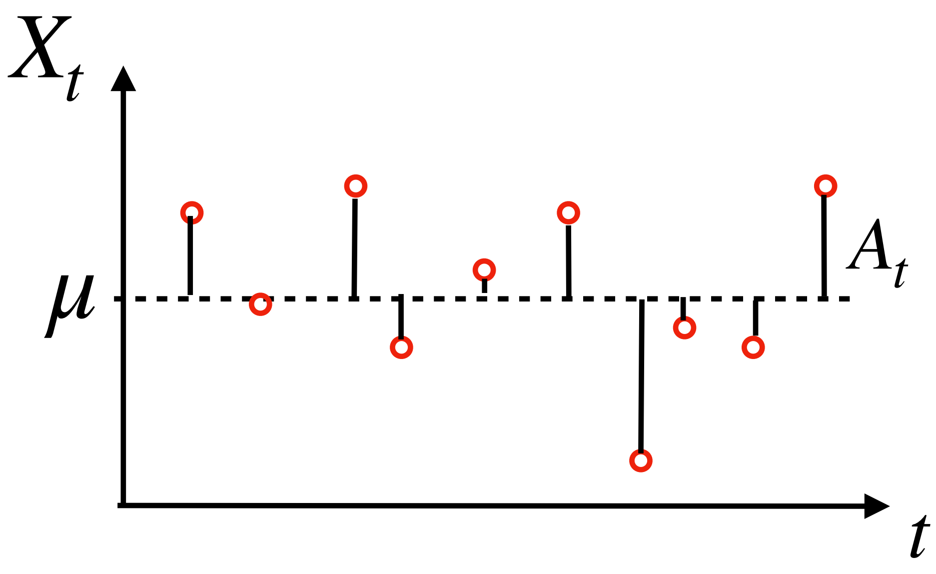
Autoregressive integrated moving average models

p d q

- General naming ARIMA(p, d, q)
- but if a hyper parameter = 0 can drop AR, I, or MA from the name
- e.g., AR(1) = ARIMA$(1,0,0)$, ARMA$(2,1)$ = ARIMA$(2,0,1)$, I(2) = ARIMA$(0,2,0)$

Special case: $p = 0,\ d = 0,\ q = 0$

$$X_t = \mu + A_t$$



White noise

- signals are generated with equal (noise) power at all frequencies ("white")
- for discrete time signals: series of random variables with zero mean & finite variance that have zero autocorrelation for non-zero lag
- thus: A_{t_1} **does not depend on** A_{t_2} **for** $t_1 \neq t_2$
- various possibilities for random distribution of A_t at given t, e.g., standard normal distribution ➜ Gaussian white noise

AR I MA

Special case: $p = 0,\ d = 0,\ q > 0$

MA models

$$X_t = \mu + A_t + \theta_1 A_{t-1} + \theta_2 A_{t-2} + \ldots + \theta_q A_{t-q}$$

- current model depends on the q past values of the noise terms $A_t, \ldots, A_{t-q}$
- noise is assumed to be normally distributed with mean zero and variance 1
- note: different sign convention for θs in literature & tools (R uses +)

> The name "moving average" is somewhat misleading because the weights, which multiply the As, need not total unity nor need that be positive. However, this nomenclature is in common use, and therefore we employ it.
> (Box & Jenkins 1976)

Example MA(2) model: $X_t = \mu + A_t + \theta_1 A_{t-1} + \theta_2 A_{t-2}$

- Mean: $\mathrm{E}[X_t] = \mu$
- Autocorrelation $\rho(1) = \dfrac{\theta_1 + \theta_1\theta_2}{1 + \theta_1^2 + \theta_2^2}$ $\rho(2) = \dfrac{\theta_2}{1 + \theta_1^2 + \theta_2^2}$ $\rho(n) = 0\ \forall n \geq 3$

➜ Autocorrelation of time series is zero for lags > q
(only approximately true for sample ACF)

MA(2) model $X_t = \mu + A_t + \theta_1 A_{t-1} + \theta_2 A_{t-2}$

Optional Slide

Mean of X_t

$$\mathrm{E}[X_t] = \mathrm{E}[\mu + A_t + \theta_1 A_{t-1} + \theta_2 A_{t-2}] = \mu + \mathrm{E}[A_t] + \theta_1 \mathrm{E}[A_{t-1}] + \theta_2 \mathrm{E}[A_{t-2}] = \mu$$

Variance of X_t

$$\begin{aligned}
\mathrm{Var}[X_t] &= \mathrm{E}[(X_t - \mu)^2] \\
&= \mathrm{E}[(A_t + \theta_1 A_{t-1} + \theta_2 A_{t-2})^2] \\
&= \mathrm{E}[A_t^2] + \theta_1^2 \mathrm{E}[A_{t-1}^2] + \theta_2^2 \mathrm{E}[A_{t-2}^2] + 2\theta_1 \cancelto{0}{\mathrm{E}[A_t A_{t-1}]} + 2\theta_2 \cancelto{0}{\mathrm{E}[A_t A_{t-2}]} + 2\theta_1\theta_2 \cancelto{0}{\mathrm{E}[A_{t-1} A_{t-2}]} \\
&= (1 + \theta_1^2 + \theta_2^2)\mathrm{E}[A_t^2] = (1 + \theta_1^2 + \theta_2^2)\mathrm{Var}[A_t]
\end{aligned}$$

Covariance for lag $n = 1$

$$\begin{aligned}
\mathrm{Cov}[X_t, X_{t+1}] &= \mathrm{E}[(X_t - \mu)(X_{t+1} - \mu)] \\
&= \mathrm{E}[(A_t + \theta_1 A_{t-1} + \theta_2 A_{t-2})(A_{t+1} + \theta_1 A_t + \theta_2 A_{t-1})] \\
&= \cancelto{0}{\mathrm{E}[A_t A_{t+1}]} + \theta_1 \mathrm{E}[A_t^2] + \theta_2 \cancelto{0}{\mathrm{E}[A_t A_{t-1}]} + \theta_1 \cancelto{0}{\mathrm{E}[A_{t-1} A_{t+1}]} + \theta_1^2 \cancelto{0}{\mathrm{E}[A_{t-1} A_t]} \\
&\quad + \theta_1\theta_2 \mathrm{E}[A_{t-1}^2] + \theta_2 \cancelto{0}{\mathrm{E}[A_{t-2} A_{t+1}]} + \theta_1\theta_2 \cancelto{0}{\mathrm{E}[A_{t-2} A_t]} + \theta_2^2 \cancelto{0}{\mathrm{E}[A_{t-2} A_{t-1}]} \\
&= \theta_1 \mathrm{E}[A_t^2] + \theta_1\theta_2 \mathrm{E}[A_{t-1}^2] = (\theta_1 + \theta_1\theta_2)\mathrm{Var}[A_t]
\end{aligned}$$

Autocorrelation for lag $n = 1$

$$\rho(1) = \frac{\mathrm{Cov}[X_t, X_{t+1}]}{\mathrm{Var}[X_t]} = \frac{(\theta_1 + \theta_1\theta_2)\mathrm{Var}[A_t]}{(1 + \theta_1^2 + \theta_2^2)\mathrm{Var}[A_t]} = \frac{\theta_1 + \theta_1\theta_2}{1 + \theta_1^2 + \theta_2^2}$$

MA(2) model $X_t = \mu + A_t + \theta_1 A_{t-1} + \theta_2 A_{t-2}$

Covariance for lag $n = 2$

$$\begin{aligned}
\mathrm{Cov}[X_t, X_{t+2}] &= \mathrm{E}[(X_t - \mu)(X_{t+2} - \mu)] \\
&= \mathrm{E}[(A_t + \theta_1 A_{t-1} + \theta_2 A_{t-2})(A_{t+2} + \theta_1 A_{t+1} + \theta_2 A_t)] \\
&= \theta_2 \mathrm{E}[A_t^2] = \theta_2 \mathrm{Var}[A_t]
\end{aligned}$$

Autocorrelation for lag $n = 2$

$$\rho(2) = \frac{\mathrm{Cov}[X_t, X_{t+2}]}{\mathrm{Var}[X_t]} = \frac{\theta_2}{1 + \theta_1^2 + \theta_2^2}$$

If $n = 3$ (or greater)

$$\begin{aligned}
\mathrm{Cov}[X_t, X_{t+3}] &= \mathrm{E}[(X_t - \mu)(X_{t+3} - \mu)] \\
&= \mathrm{E}[(A_t + \theta_1 A_{t-1} + \theta_2 A_{t-2})(A_{t+3} + \theta_1 A_{t+2} + \theta_2 A_{t+1})] \\
&= 0
\end{aligned}$$

$$\rho(3) = \frac{\mathrm{Cov}[X_t, X_{t+3}]}{\mathrm{Var}[X_t]} = 0$$

Special case: $p > 0,\ d = 0,\ q = 0$

AR models

$$X_t = \delta + \phi_1 X_{t-1} + \phi_2 X_{t-2} + \ldots + \phi_p X_{t-p} + A_t$$

- current model depends on the p past values of the time series
- white noise random term

➜ partial auto-correlation of time series is zero for lags > p

(only approximately true for sample PACF)

Example AR(1) model: $X_t = \delta + \phi_1 X_{t-1} + A_t$

- Mean: $\mathrm{E}[X_t] = \mu = \dfrac{\delta}{1 - \phi_1}$
- Variance: $\mathrm{Var}[X_t] = \mathrm{Var}[A_t]\dfrac{1}{1 - \phi_1^2}$
- Autocorrelation $\rho(n) = \phi_1^n$

ACF decays slowly towards zero for $|\phi_1| < 1$

Case $|\phi_1| = 1$ (unit root)

Stationary implies that the ACF goes to zero for $n \to \infty$, Hence: $|\phi_1| = 1$ is non-stationary

➜ Random walk $X_t = X_{t-1} + A_t$

We can remove a unit root by differencing: $\Delta X_t \equiv X_t - X_{t-1} = A_t$ ➜ ΔX_t is stationary

Case $|\phi_1| > 1$: explosive process, $|X_t|$ increases without bounds with time

AR(1) model $X_t = \delta + \phi_1 X_{t-1} + A_t$

Optional Slide

Mean of X_t $\quad \mathrm{E}[X_t] = \mathrm{E}[\delta + \phi_1 X_{t-1} + A_t] = \delta + \phi_1 \mathrm{E}[X_{t-1}] + \mathrm{E}[A_t] = \delta + \phi_1 \mathrm{E}[X_t]$

$$\rightarrow \mathrm{E}[X_t] = \mu = \frac{\delta}{1 - \phi_1} \qquad \text{i.e., } \delta = \mu - \mu\phi_1$$

Variance of X_t

$$\begin{aligned}
\mathrm{Var}[X_t] &= \mathrm{E}[(X_t - \mu)^2] = \mathrm{E}[(\delta + \phi_1 X_{t-1} + A_t - \mu)^2] = \mathrm{E}[(\phi_1 X_{t-1} + A_t - \mu\phi_1)^2] \\
&= \phi_1^2 \mathrm{E}[X_{t-1}^2] + \mathrm{E}[A_t^2] + \mu^2\phi_1^2 + 2\phi_1 \mathrm{E}[X_{t-1}A_t] - 2\mu\phi_1^2 \mathrm{E}[X_{t-1}] - 2\mu\phi_1 \mathrm{E}[A_t] \\
&= \phi_1^2 \mathrm{E}[X_t^2] + \mathrm{Var}[A_t] + \mu^2\phi_1^2 + 2\phi_1 \mathrm{E}[X_{t-1}]\cancelto{0}{\mathrm{E}[A_t]} - 2\mu^2\phi_1^2 - 2\mu\phi_1 \cancelto{0}{\mathrm{E}[A_t]} \\
&= \phi_1^2 (\mathrm{E}[X_t^2] - \mu^2) + \mathrm{Var}[A_t] \\
&= \phi_1^2 \, \mathrm{Var}[X_t] + \mathrm{Var}[A_t] \qquad \rightarrow \mathrm{Var}[X_t] = \frac{1}{1 - \phi_1^2} \mathrm{Var}[A_t]
\end{aligned}$$

Covariance for lag $n > 0$

$$\begin{aligned}
\mathrm{Cov}[X_t, X_{t+n}] &= \mathrm{E}[(X_t - \mu)(X_{t+n} - \mu)] = \mathrm{E}[(\phi_1 X_{t-1} + A_t - \mu\phi_1)(\phi_1 X_{t+n-1} + A_{t+n} - \mu\phi_1)] \\
&= \phi_1^2 \mathrm{E}[X_{t-1} X_{t+n-1}] - \mu\phi_1^2 \mathrm{E}[X_{t-1}] + \phi_1 \mathrm{E}[A_t X_{t+n-1}] - \mu\phi_1^2 \mathrm{E}[X_{t+n-1}] + \mu^2\phi_1^2 \\
&= \phi_1^2 (\mathrm{E}[X_{t-1} X_{t+n-1}] - \mu^2) + \phi_1 \mathrm{E}[A_t X_{t+n-1}] = \phi_1^2 \mathrm{Cov}[X_t, X_{t+n}] + \phi_1 \mathrm{E}[A_t X_{t+n-1}]
\end{aligned}$$

$$\rightarrow \mathrm{Cov}[X_t, X_{t+n}] = \frac{\phi_1^n}{1 - \phi_1^2} \mathrm{Var}[A_t]$$

$$\begin{aligned}
\mathrm{E}[A_t X_{t+n-1}] &= \phi_1 \mathrm{E}[A_t X_{t+n-2}] = \phi_1^2 \mathrm{E}[A_t X_{t+n-3}] = \ldots \\
&= \phi_1^{n-1} \mathrm{E}[A_t X_t] = \phi_1^{n-1} \mathrm{E}[A_t A_t] = \phi_1^{n-1} \mathrm{Var}[A_t]
\end{aligned}$$

Autocorrelation for lag n $\quad \rho(n) = \dfrac{\mathrm{Cov}[X_t, X_{t+n}]}{\mathrm{Var}[X_t]} = \phi^n$

AR I MA

Simple

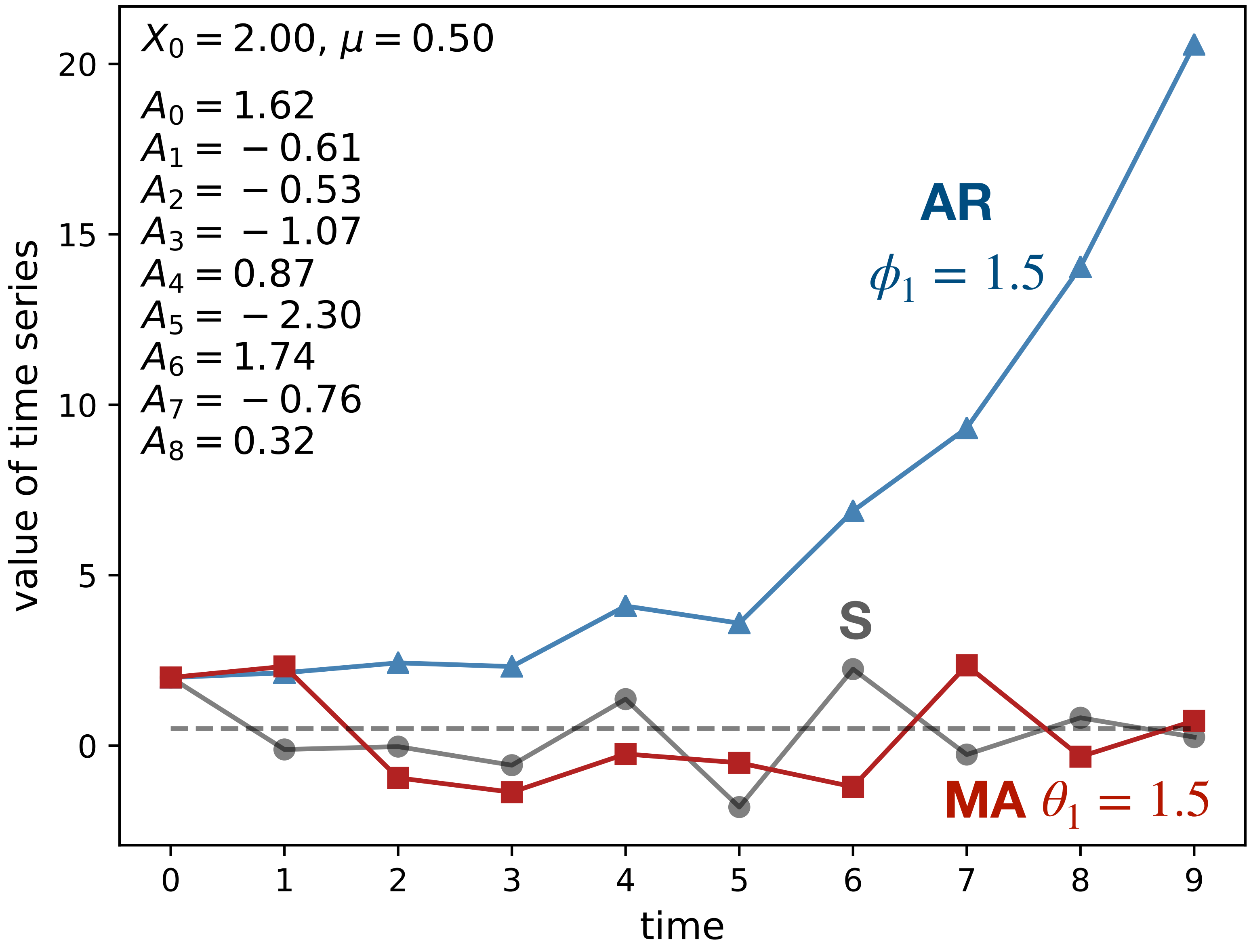
$$X_t = \mu + A_t$$

AR(1)

$$X_t = \mu + A_t + \phi_1(X_{t-1} - \mu)$$

MA(1)

$$X_t = \mu + A_t + \theta_1 A_{t-1}$$



Model Identification with ARIMA

Step 1: Stationarity & Seasonality

- Is time series stationary?
  - plot time series (constant mean & variance), check ACF tapers off towards 0
  - Augmented Dickey-Fuller (ADF) test
  - differencing (order d) or fit trend and subtract/divide
- Is there seasonality?
  - autocorrelation plot or seasonal sub-series plot
  - once found and period known ➜ parameters for **seasonal ARIMA**

    additional parameters P, D, Q for seasonal AR, I, and MA terms
  - or use seasonal differencing to remove seasonality

Step 2: Find *p* and *q*

- autocorrelation plot ➜ tells us q (MA part)
- partial autocorrelation plot ➜ tells us p (AR part)
- typically check plots to find (small) lags with <u>statistically significant</u> (95% conf. intervals) autocorrelations or partial autocorrelations

Model Identification with ARIMA

Shape of autocorrelation function	Indicated model
Decaying to zero, may or may not alternate between+ and -	AR model, use partial autocorrelation plot to identify *p*
One or more spikes, rest are essentially zero	MA model, order *q* identified as last non-zero lag
Decay starting after a few lags	Mixed AR and MA model
All close to zero	Data is essentially random
High values at fixed intervals	Include seasonal autoregressive term
No decay to zero	Data is not stationary

Model Estimation with ARIMA

- non-linear least squares fitting or maximum likelihood methods
- use software! (e.g., statsmodels in Python)

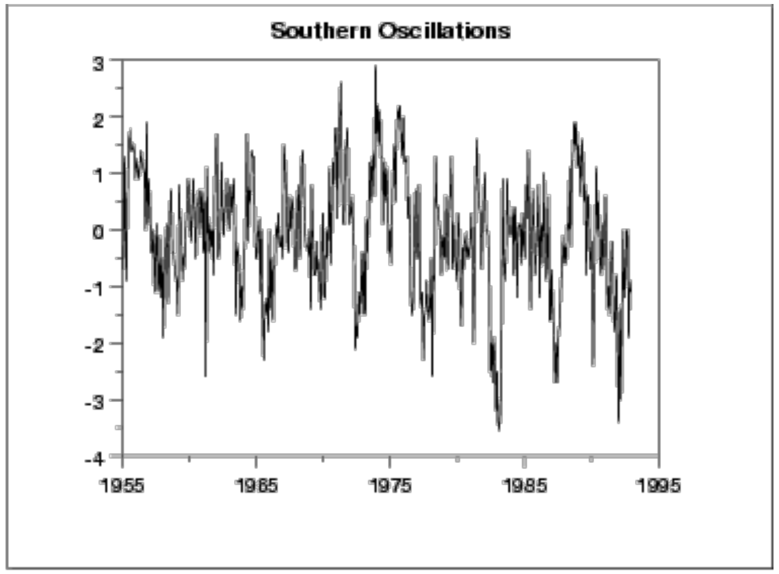
https://towardsdatascience.com/machine-learning-part-19-time-series-and-autoregressive-integrated-moving-average-model-arima-c1005347b0d7

Model Validation with ARIMA

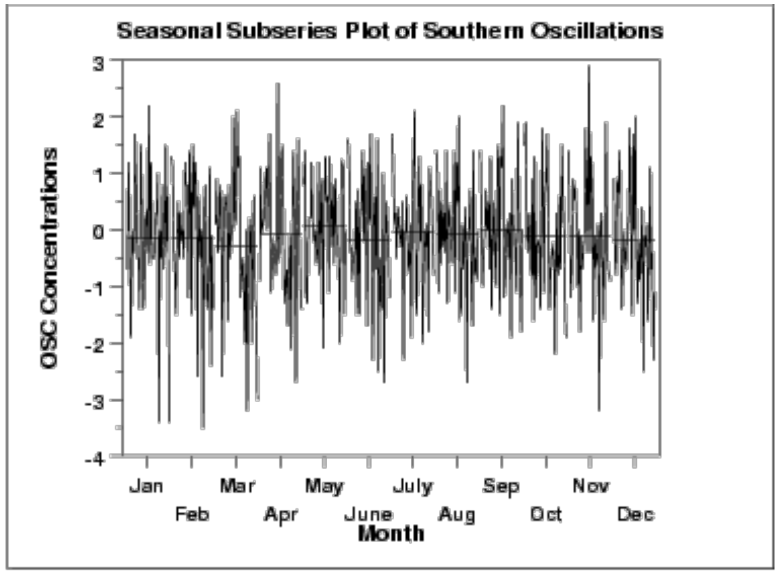
- as before: residuals should be white noise
- 4-plot, autocorrelation plot
- Box-Ljung statistical test

Example 1 - Atmospheric Oscillations

time series

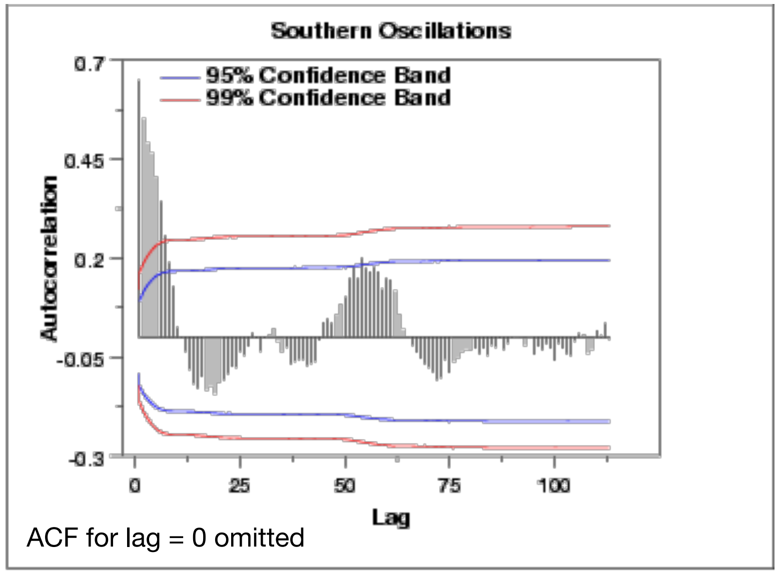


seasonal subseries plot

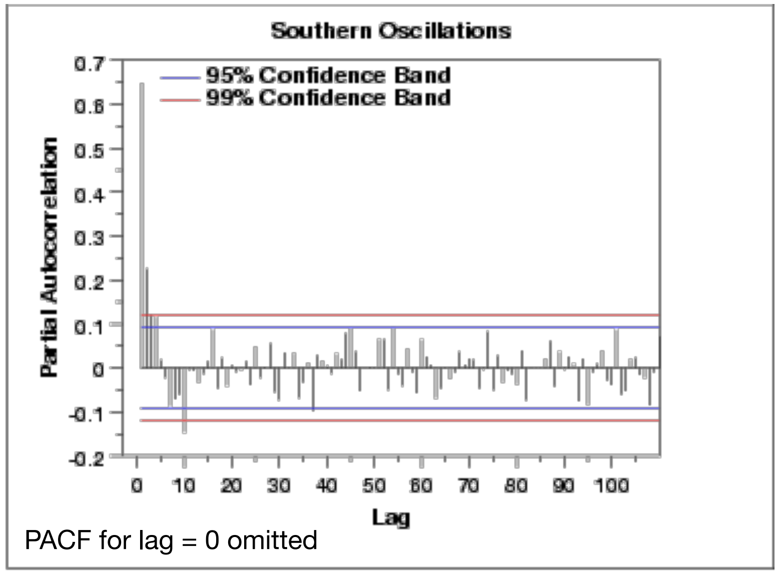


?

ACF



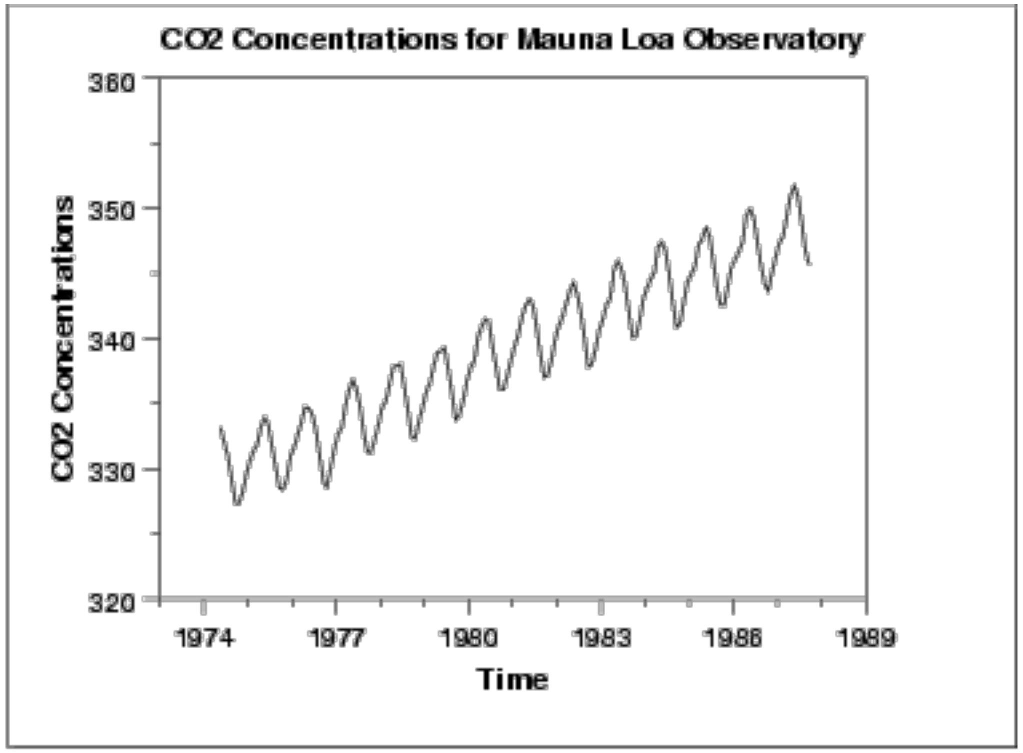
PACF

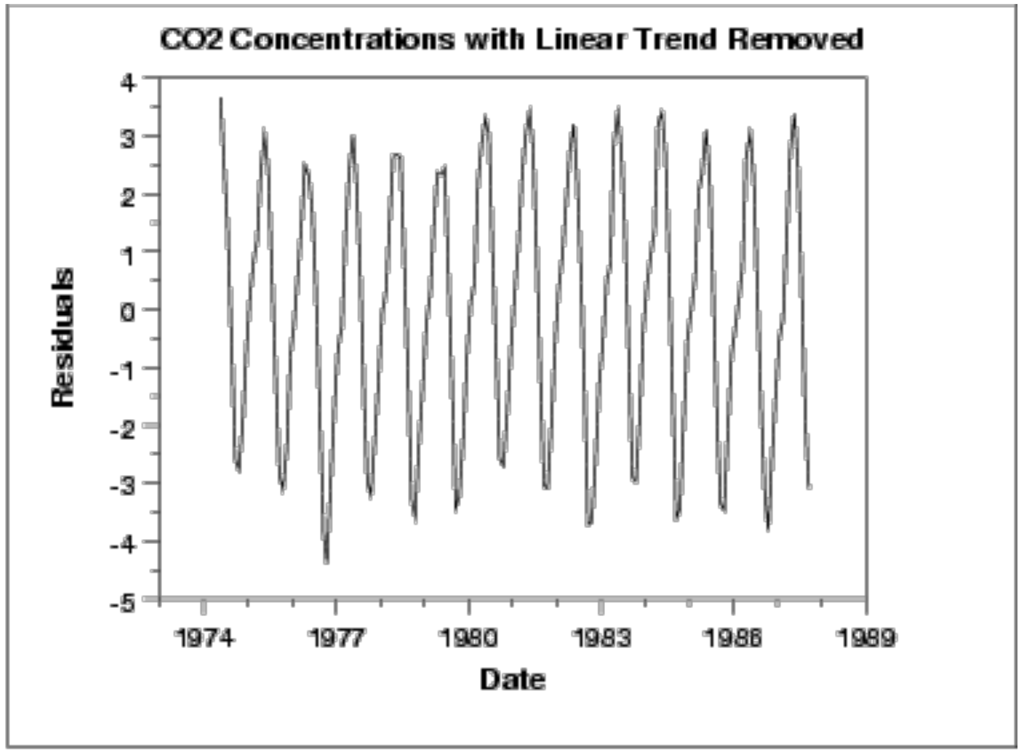


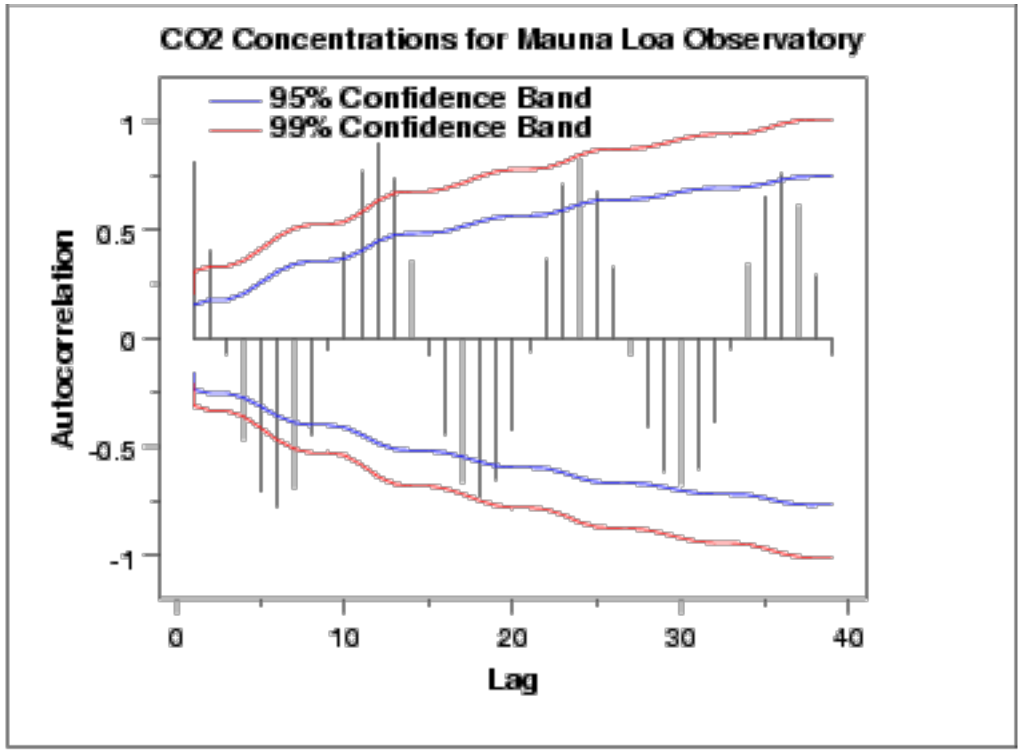
The shown model is

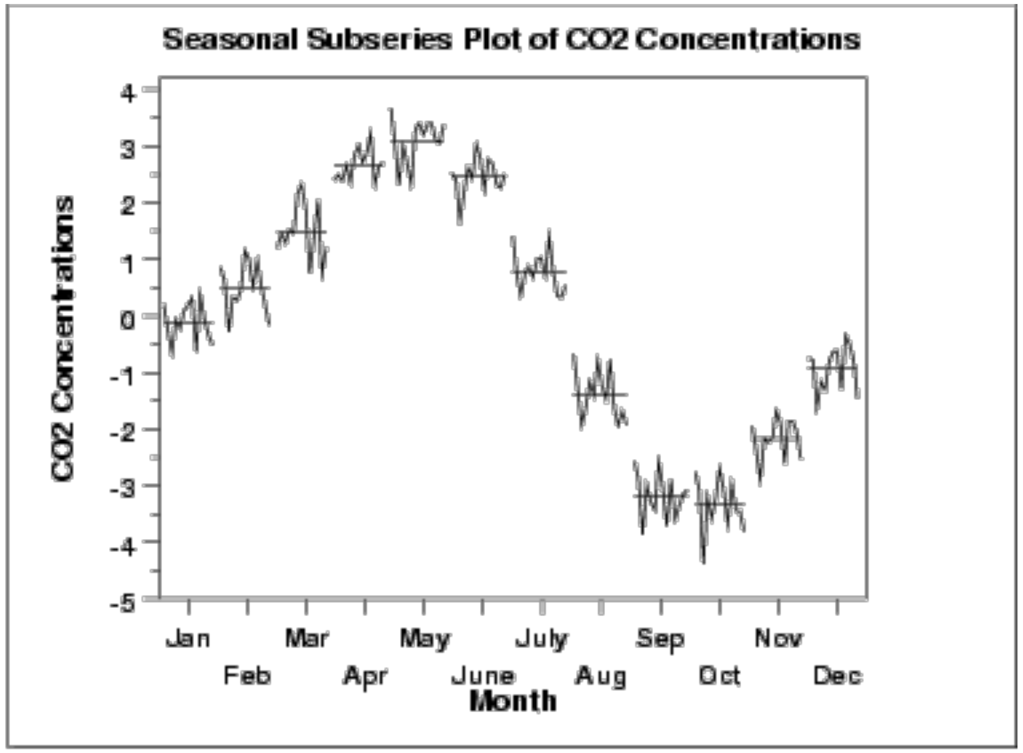
A) stationary, with a seasonal dependence, MA(7) appropriate
B) stationary, no seasonal dependence, AR(2) appropriate
C) non-stationary, no seasonal dependence, ARI(2, 1) appropriate
D) stationary, no seasonal dependence, ARMA(1,1) appropriate

Example 2 - Monthly CO_2 concentrations





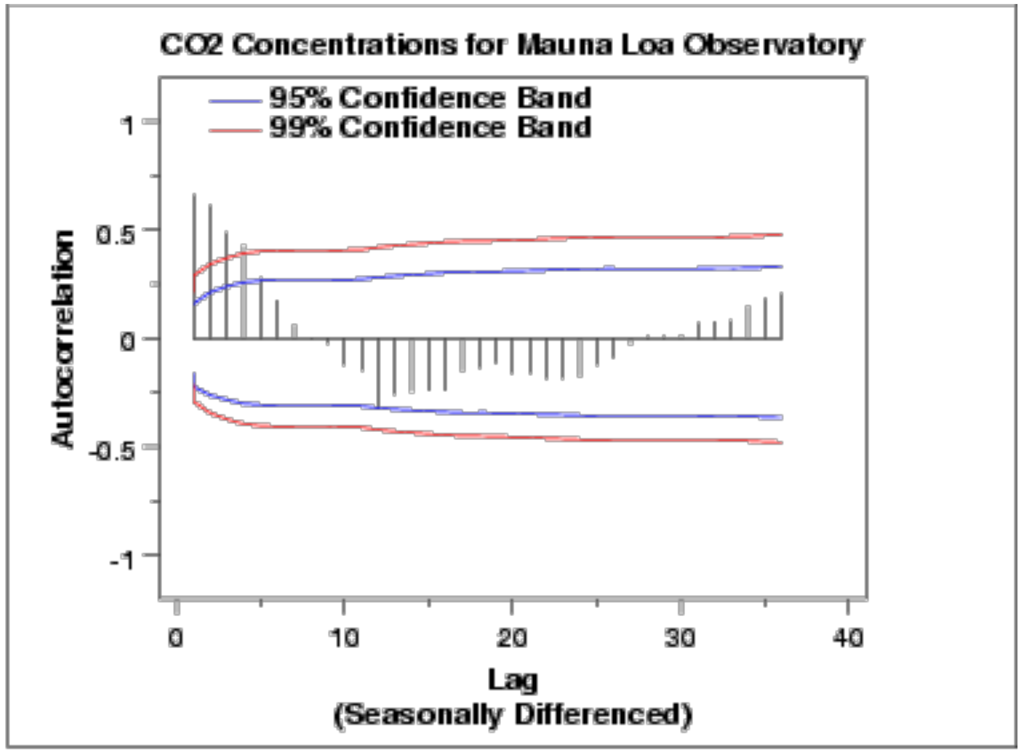




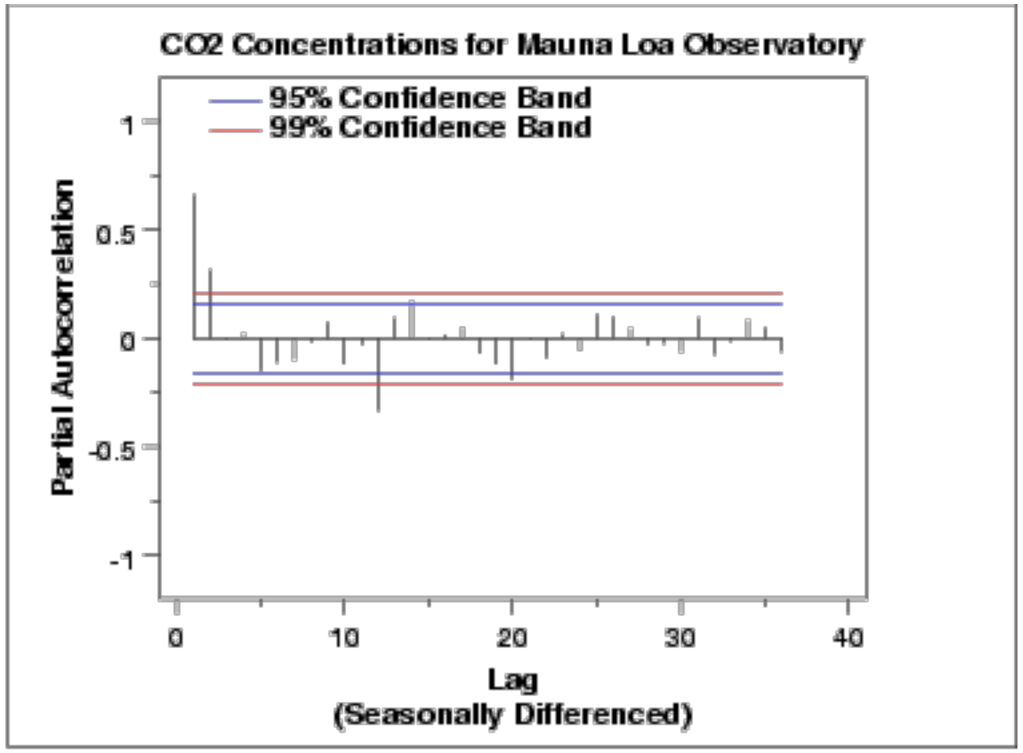
Example 2 - Monthly CO_2 concentrations

- de-trend to remove obvious increase with time
- include a 12-lag seasonal AR or MA term in seasonal ARIMA
- apply seasonal difference to check autocorrelation and partial autocorrelation

➜ after de-trending / removal of seasonality



autocorrelation

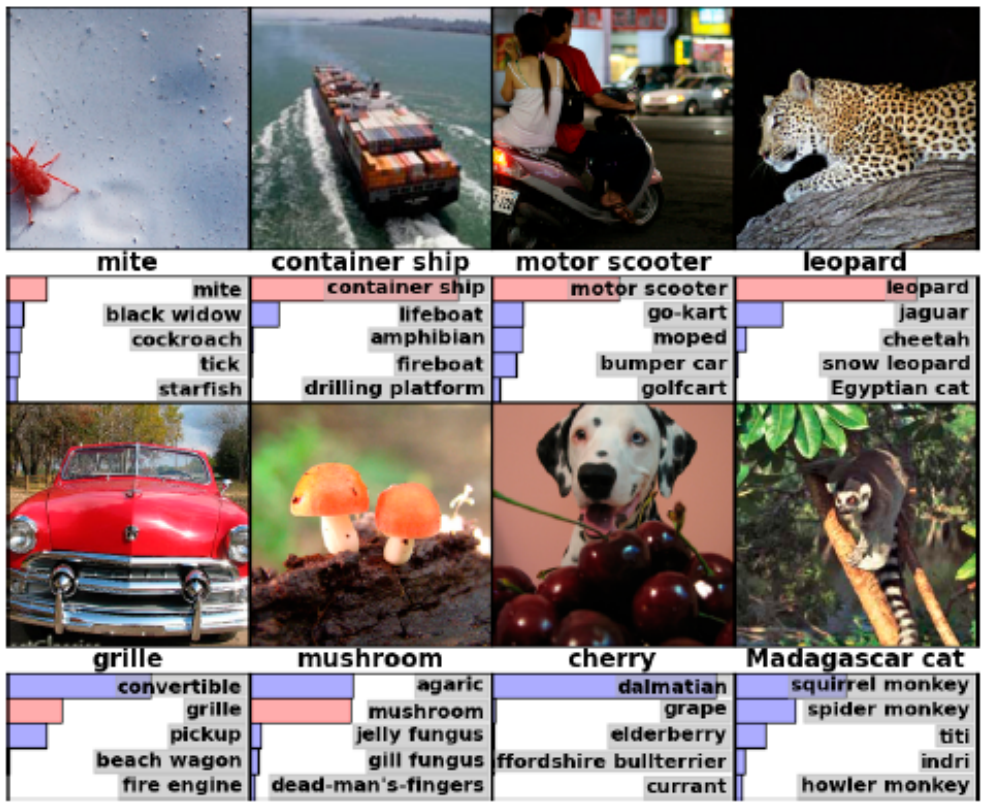


partial autocorrelation

- partial autocorr. plot suggests AR(2) model
- note: peak at lag=12 even after seasonal differencing

Artificial neural networks

- Powerful class of machine learning models, well suited for many tasks
  - image recognition, gaming, speech recognition, natural language processing, self-driving, …



AlexNet (CNN) won the 2012 ImageNet competition

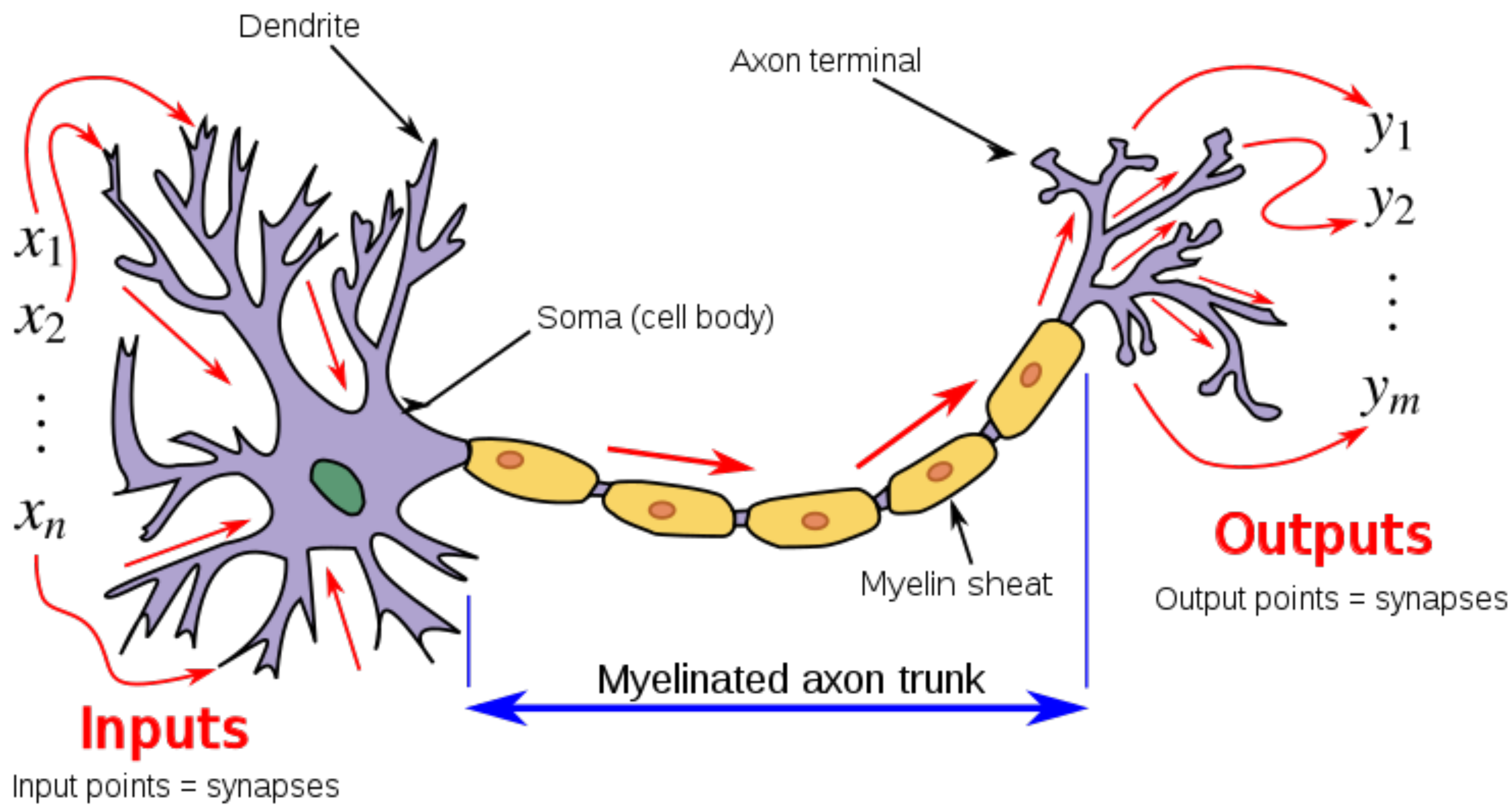
AlphaGo vs Lee Sedol (Seoul, 2016)
4:1 for AlphaGo

- Can reach Human performance and even exceed it!

- Simplified model of biological neural networks ➜ neuroscience

Learning from the Brain

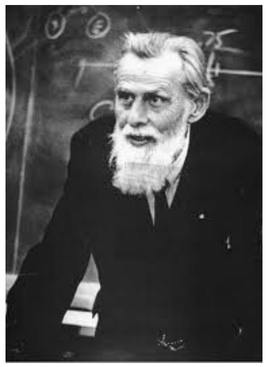
- Brain = complex network of neurons; how it functions still not well understood
- Biological neurons are smallest units

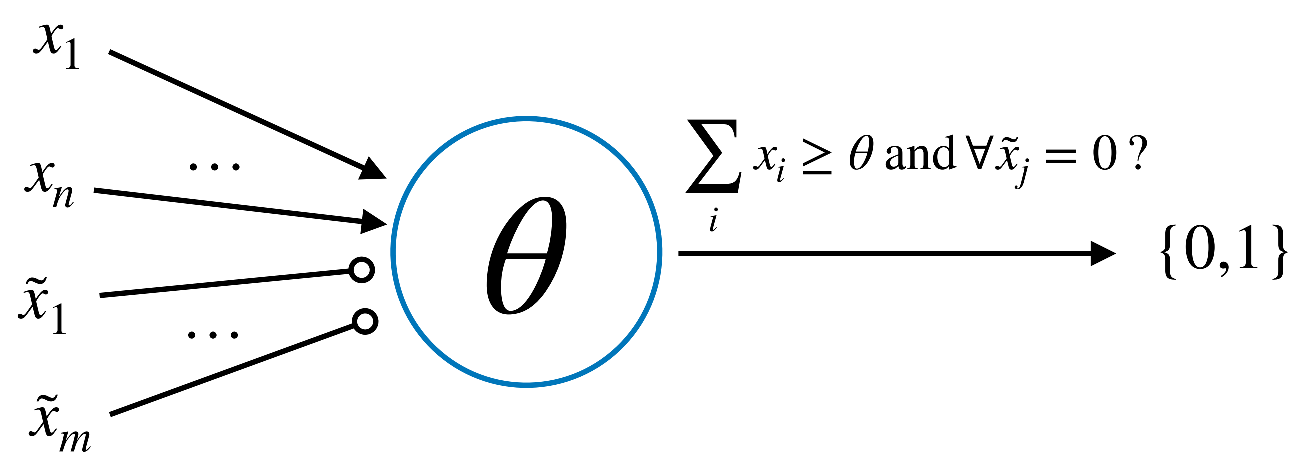


- Idea: Mimic how the brain works to build better ML models
- requires substantial abstraction, i.e., biology only used as a guide

McCulloch and Pitts (MCP) Neuron

- McCulloch & Pitts 1943





<u>Simple model of a neuron that **implements boolean functions**</u>

- multiple binary input and a single binary output
- input of two types: excitatory x_i and inhibitory $\tilde{x}_j$

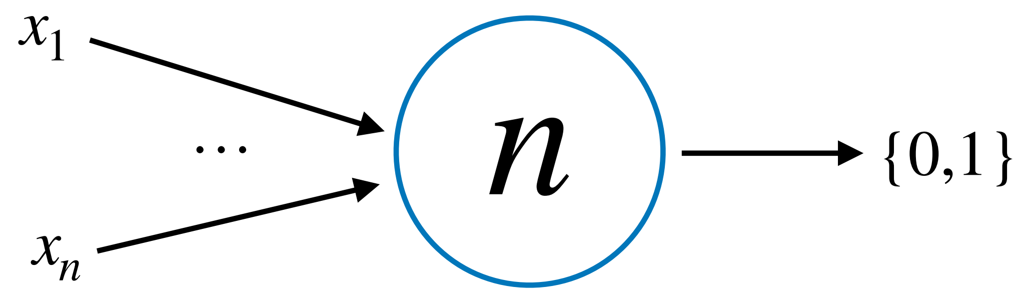
<u>How is the output calculated?</u>

- if any of the inhibitory inputs ($\tilde{x}_j$) is 1, output 0
- otherwise, sum all the excitatory inputs (x_i) and compare with threshold θ
    - if the sum is $\geq \theta$ output 1, otherwise output 0

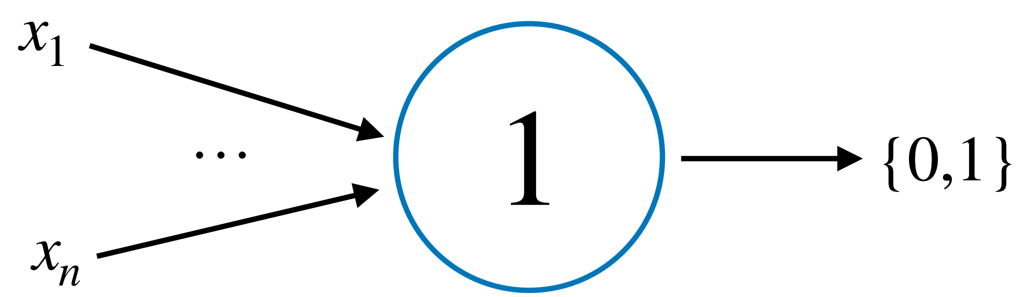
McCulloch and Pitts (MCP) Neuron

Example Boolean Functions:

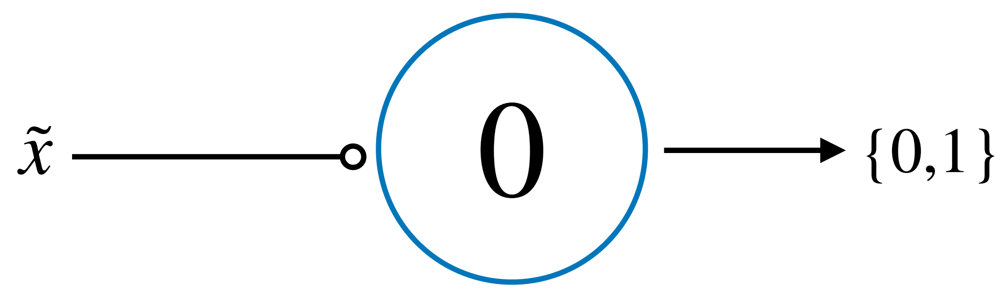
AND $x_1 \wedge x_2 \wedge \ldots \wedge x_n$



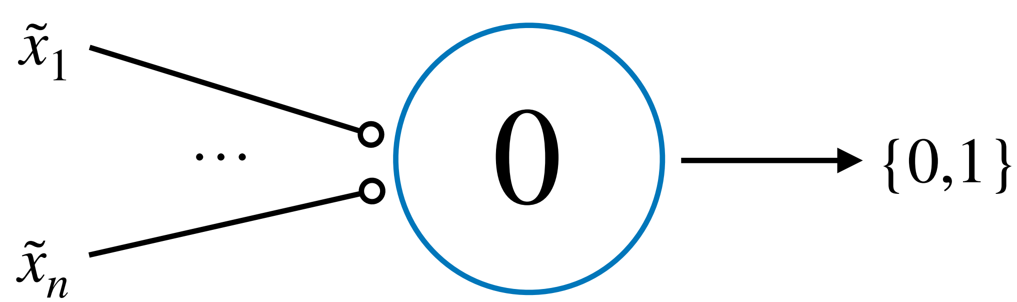
OR $x_1 \vee x_2 \vee \ldots \vee x_n$



NOT $\neg x$



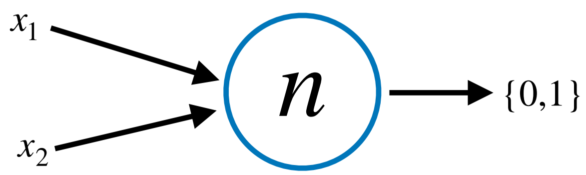
NOR $\neg x_1 \wedge \ldots \wedge \neg x_n$

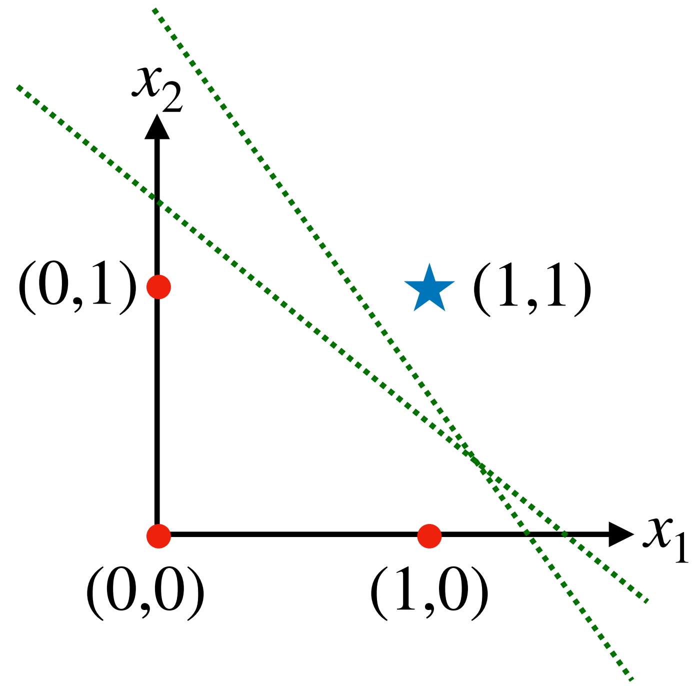


McCulloch and Pitts (MCP) Neuron

Geometric interpretation

AND $x_1 \wedge x_2$





★ output is 1

● output is 0

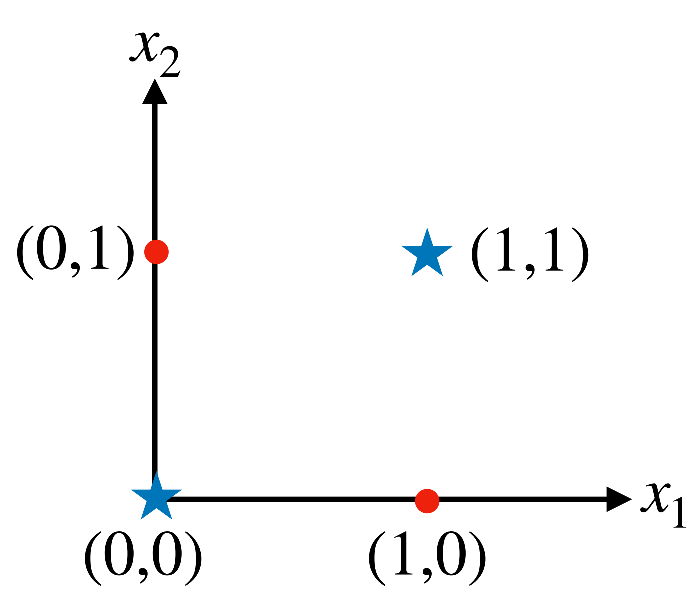
Note: $x_i \in \{0,1\}$

- encodes a linear separation in (discrete) feature space
- (hyper-)plane in 3+ dimensions, but not unique

What about XOR?

$$x_1 \oplus x_2$$

x_1	x_2	$x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0



- Cannot be represented by single neuron ➜ network
- *Networks* made from MCP neurons are functionally complete, i.e., can represent any boolean function
- Proof: MCP can calculate {AND, NOT} and every boolean formula can be build from these

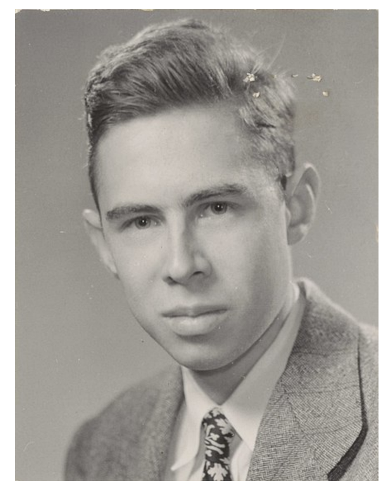
Perceptron

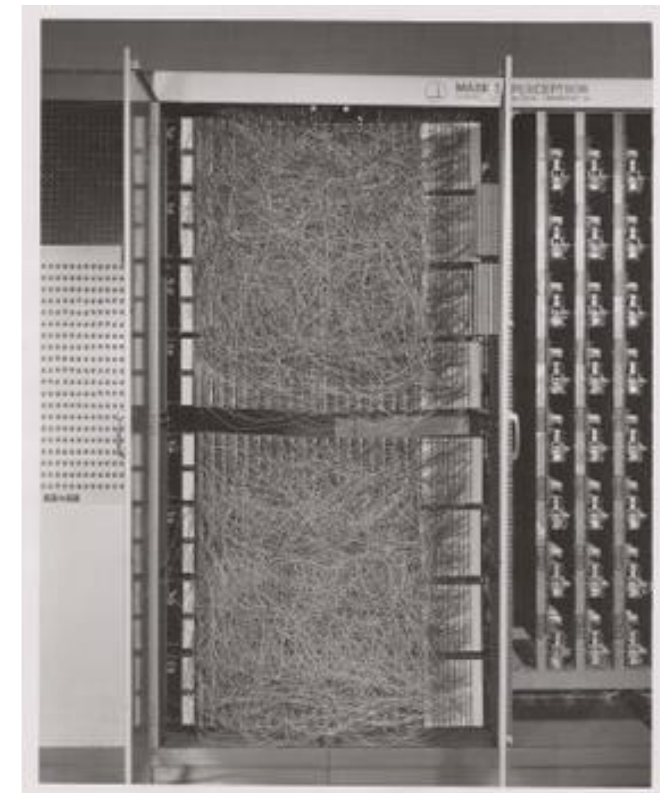
Limitations of MCP:

- Only boolean (binary) inputs
- Threshold is hand-coded for each function
- All inputs have similar importance in general

Perceptron:

- invented by Frank Rosenblatt at the Cornell Aeronautical Laboratory in 1958
- initially designed as a machine for image recognition





Mark I Perceptron machine

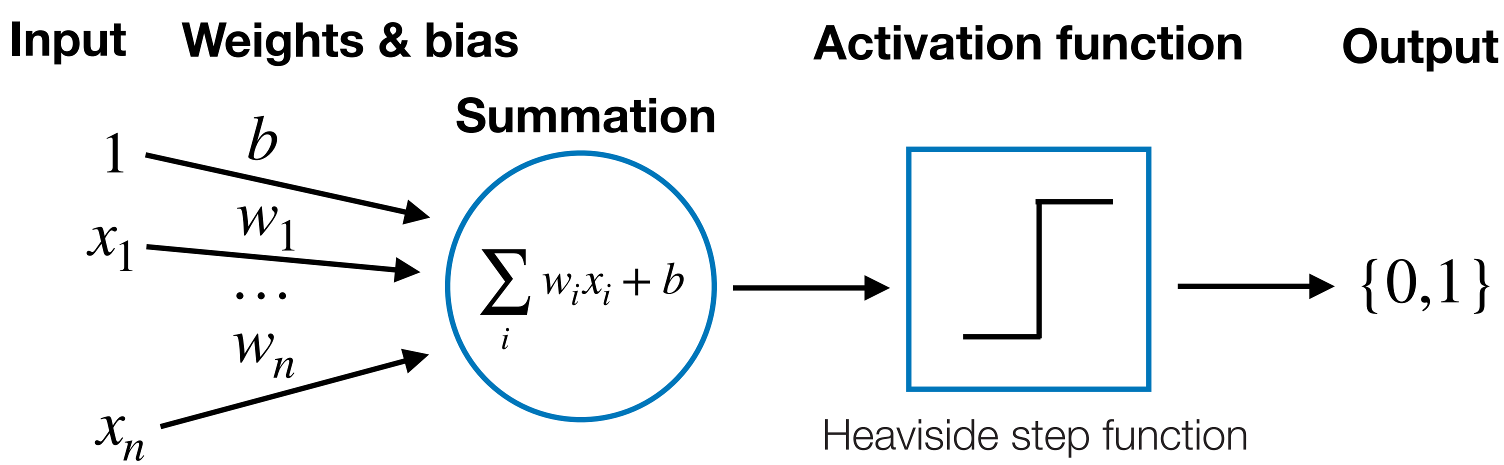
- in the modern sense: binary classifier

- Calculates $f : R^n \rightarrow \{0,1\}$

$$f(\mathbf{x}; \mathbf{w}, b) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

- n-dim real input values $\mathbf{x} \in R^n$, single binary output value
- weights $\mathbf{w} \in R^n$
- bias b

Perceptron

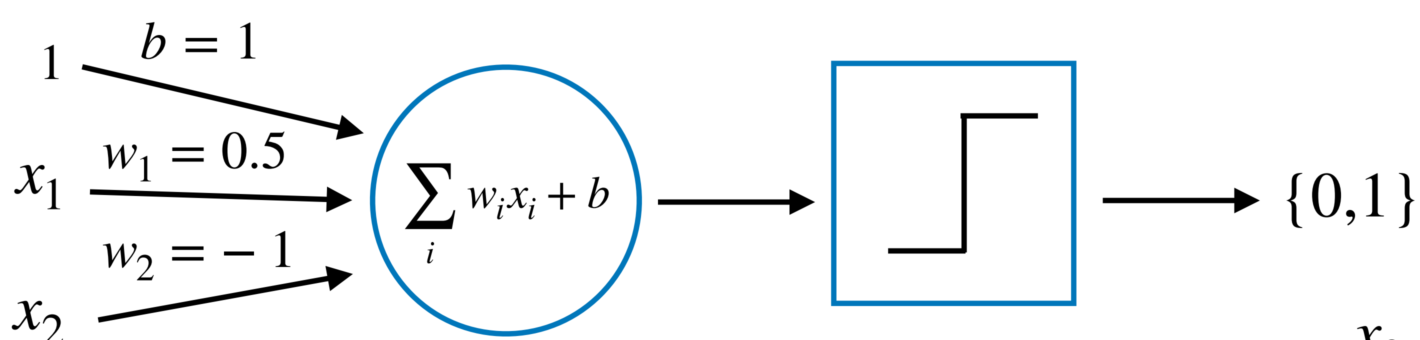


Heaviside step function

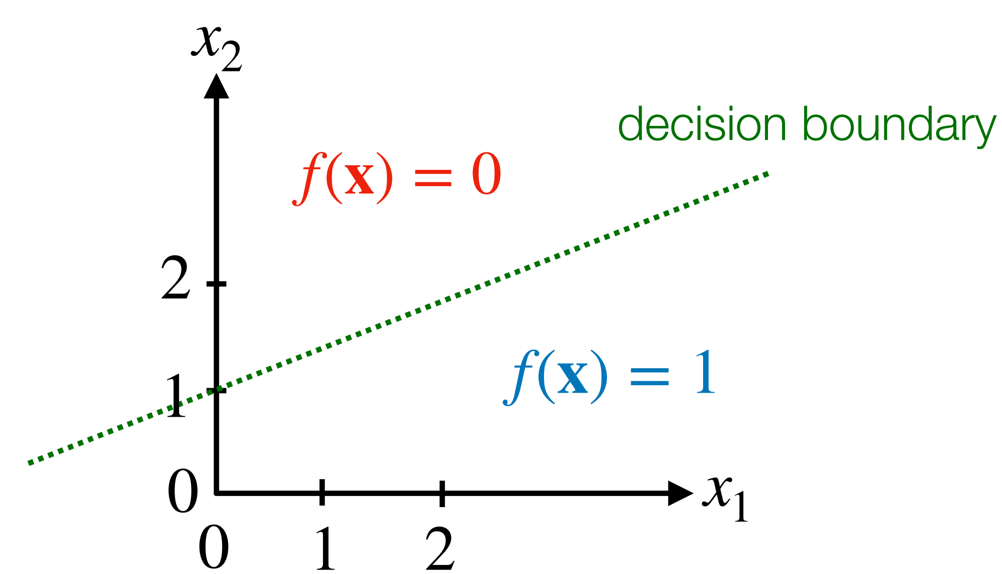
$$\Theta(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

- Can be used as a linear classifier

<u>Example</u>

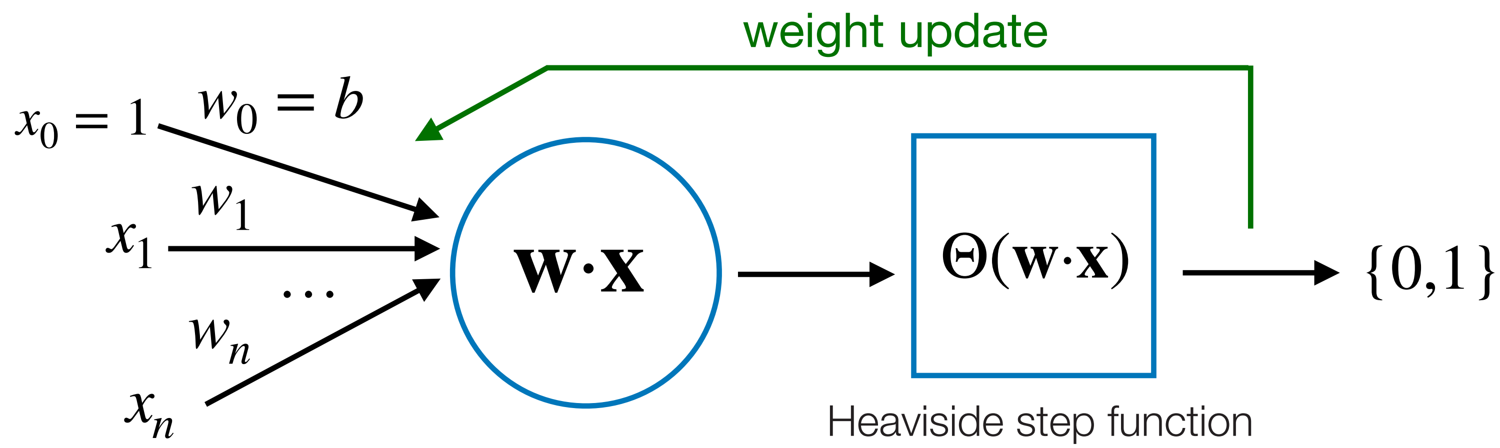


$$\begin{aligned} f(\mathbf{x}; \mathbf{w}, b) = 1 &\leftrightarrow w_1 x_1 + w_2 x_2 + b > 0 \\ &\leftrightarrow 0.5 x_1 - x_2 + 1 > 0 \\ &\leftrightarrow x_2 < 0.5 x_1 + 1 \end{aligned}$$



- weights and bias define separating (hyper-)plane!

Training of a single Perceptron



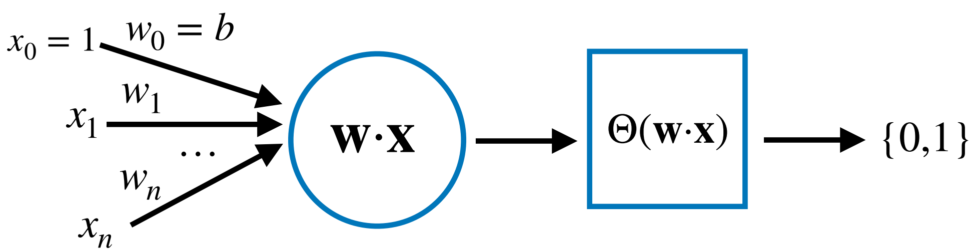
Algorithm: Perceptron Learning Algorithm

$P \leftarrow inputs \quad with \quad label \quad 1;$
$N \leftarrow inputs \quad with \quad label \quad 0;$
Initialize $\mathbf{w}$ randomly;
while $!convergence$ **do**
  Pick random $\mathbf{x} \in P \cup N$;
  if $\mathbf{x} \in P \quad and \quad \mathbf{w.x} < 0$ **then**
    $w = w + \eta x$;
  end
  if $\mathbf{x} \in N \quad and \quad \mathbf{w.x} \geq 0$ **then**
    $w = w - \eta x$;
  end
end
//the algorithm converges when all the inputs are classified correctly

η is learning rate (hyperparameter)

- Supervised learning algorithm
- need training data with class labels
- trains perceptron to be a linear classifier

- Perceptron can also represent logic gates if given binary input



Example:

OR $x_1 \vee x_2$

$$w_0 = -0.5, w_1 = 1, w_2 = 1$$

e.g., $f(\mathbf{x} = (0,1)) = \Theta(w_0 + 0w_1 + 1w_2) = \Theta(0.5) = 1$

NOT $\neg x$

$$w_0 = 0.5, w_1 = -1$$

- $f(x = 0) = \Theta(w_0 + 0w_1) = \Theta(0.5) = 1$
- $f(x = 1) = \Theta(w_0 + 1w_1) = \Theta(0.5 - 1) = 0$

Q: How would you implement an AND gate? Is the answer unique?

Q: Can all logical functions be implemented by a perceptron?

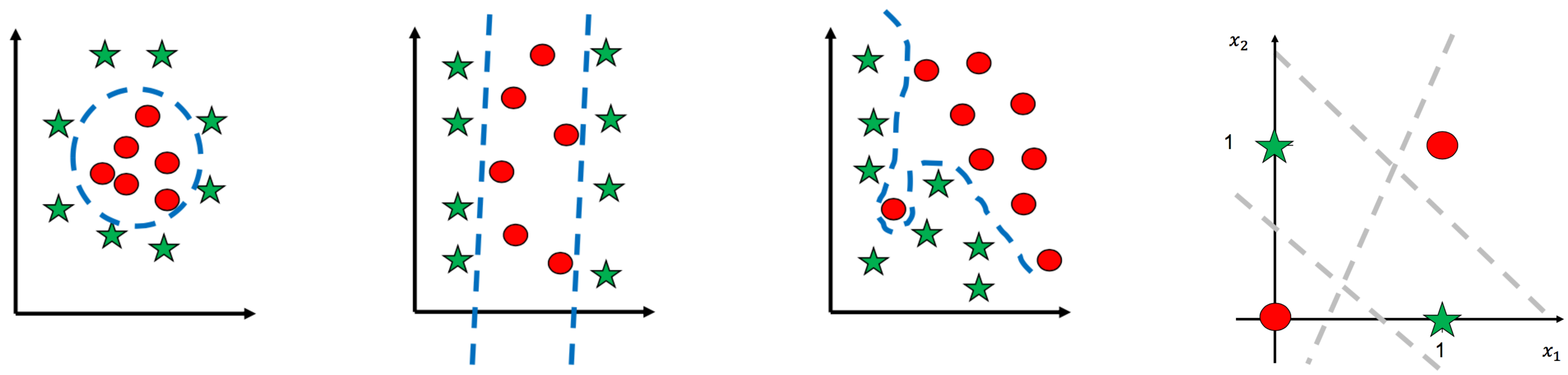
Perceptron & MCP neuron summary

- Perceptrons are linear classifiers, i.e., have a linear decision boundary (hyperplane)
- decision boundary given by weights & bias
- Perceptron can emulate MCP neuron, i.e., more powerful
- Both Perceptron & MCP neuron can implement many logical functions, but not all (XOR)
- Perceptron can be trained on labeled data (supervised learning)

<u>Unsolvable example problems:</u>

non-linear decision boundaries

disconnected boundaries

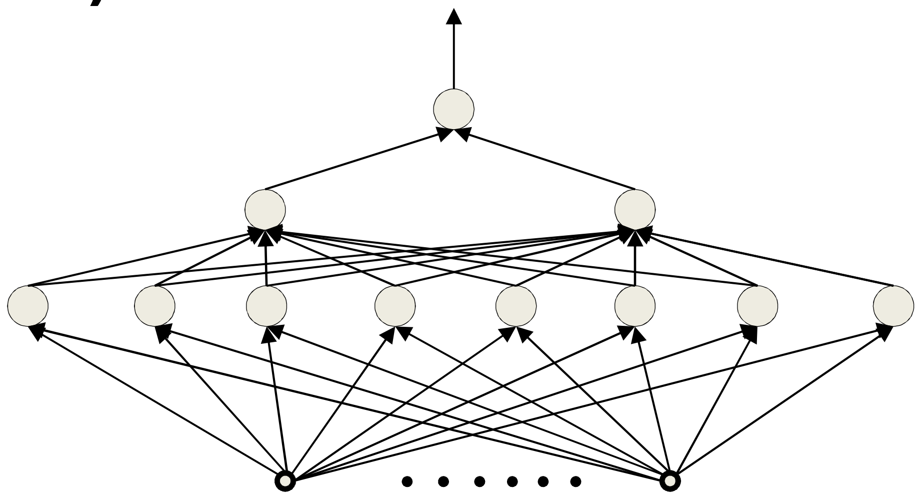


Option A: kernel trick ➜ non-linear classifier (e.g., SVM)

Option B: use a network of neurons

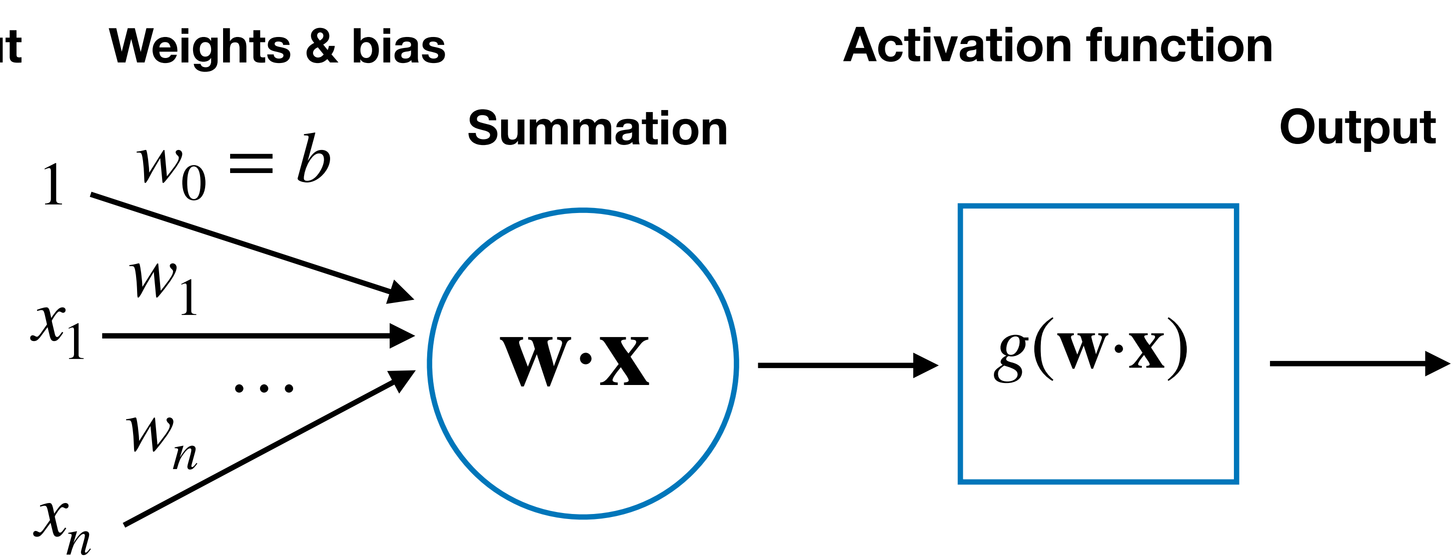
The multi-layer perceptron (MLP)

- A *fully connected* (dense) *feedforward* network of *artificial neurons*
- Many different architectures, e.g., how deep? how many nodes?



Definitions:

- *artificial neuron*: generalization of perceptron to allow for any activation function g



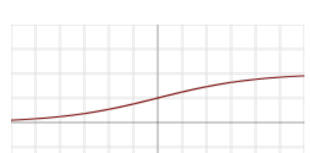
- Many activation functions exist
- choice of activation function is hyperparameter
- note the output range of given activation function

Heaviside step function

$$\Theta(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

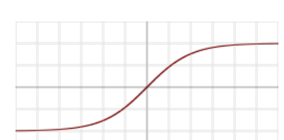
Sigmoid function

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



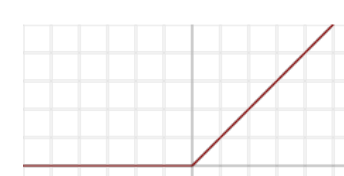
Hyperbolic tangent

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



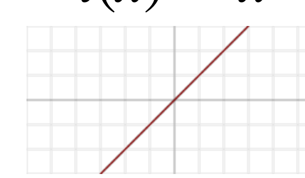
Rectified-Linear Unit (ReLU)

$$r(x) = \max(0, x)$$



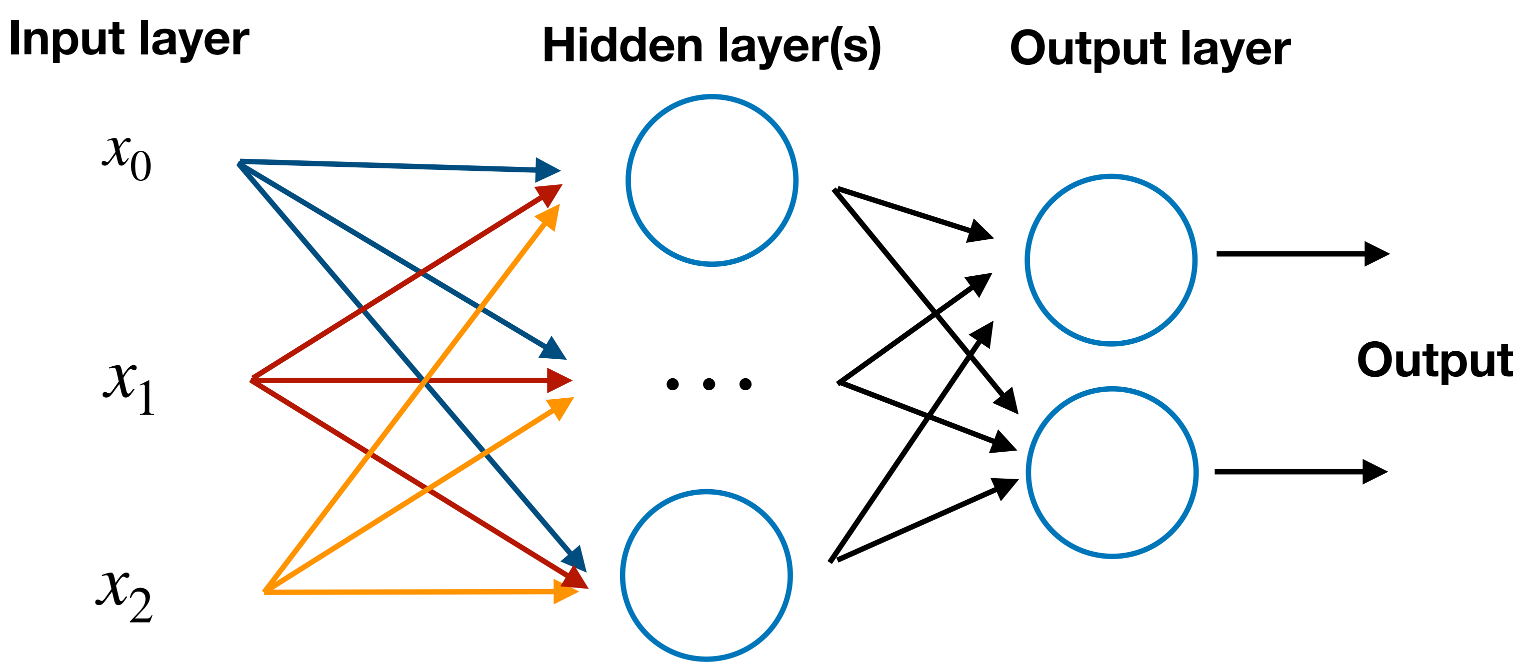
Linear activation

$$l(x) = x$$



Definitions (cont'd):

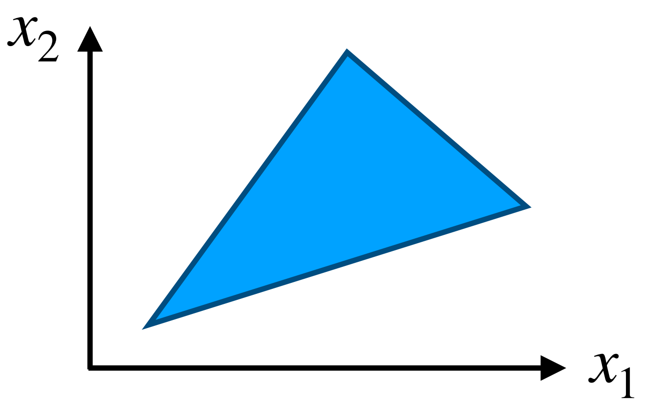
- *fully connected*: each node from one layer connected to node from next layer
- *feedforward*: no cycles, no backward connections etc.
- *depth*: longest path from input layer to output layer
- *deep network*: depth 3 or greater

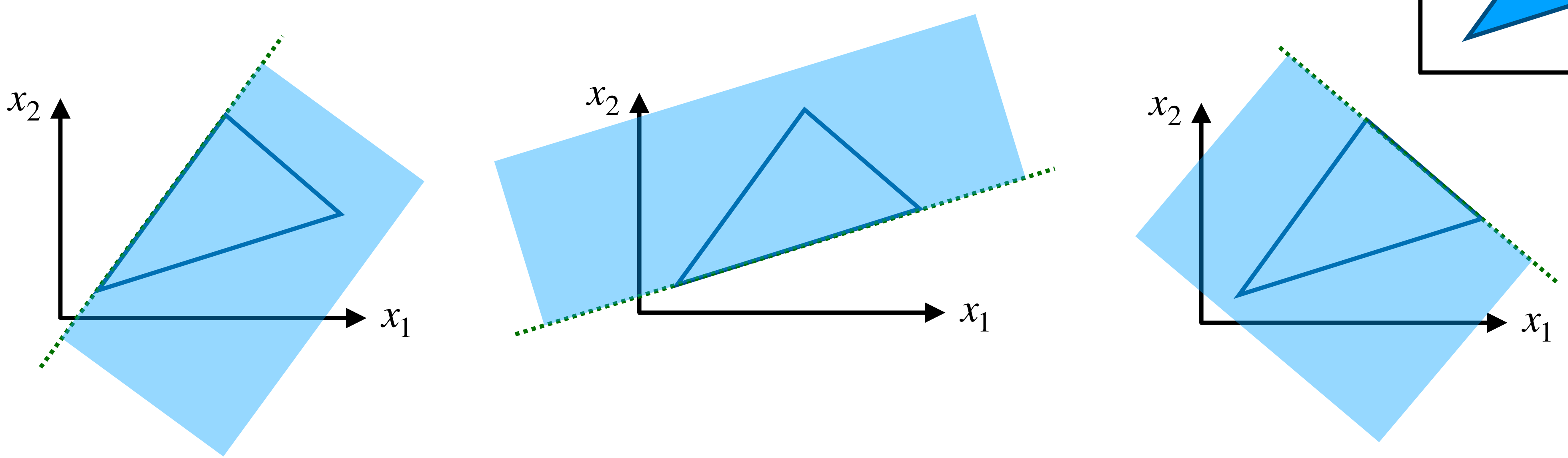


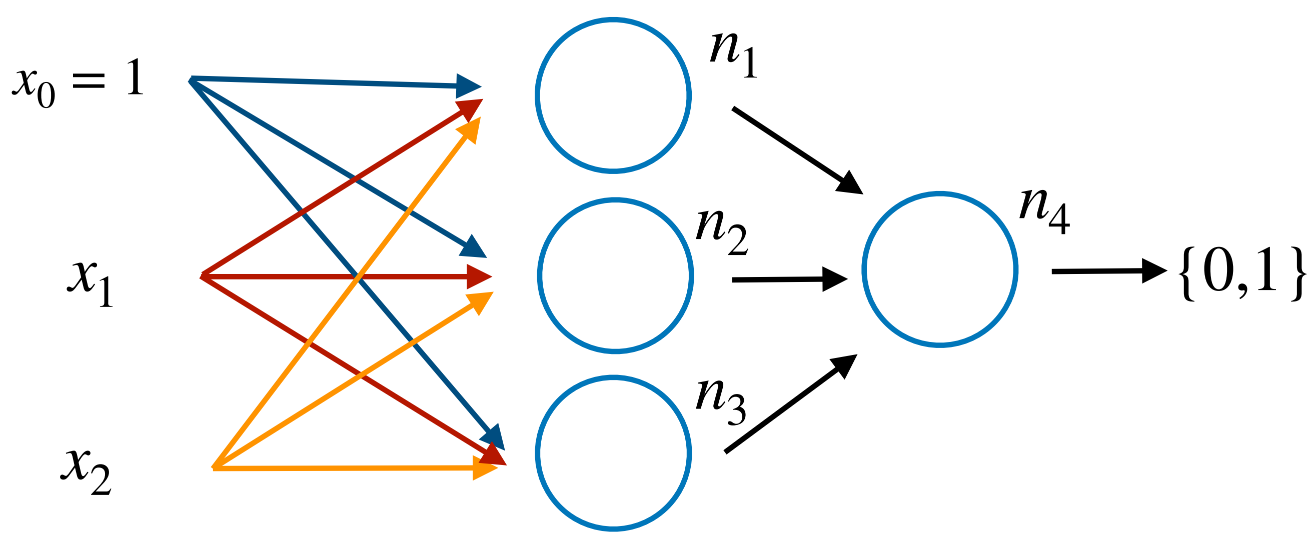
- In Example: depth = 2
- #layers = depth + 1

- We already saw that networks of MCP neurons (and thus perceptrons) can represent any boolean function ➜ *Universal boolean machines*

- They are also *Universal classifiers*
  e.g., how to represent decision boundary shown in blue?





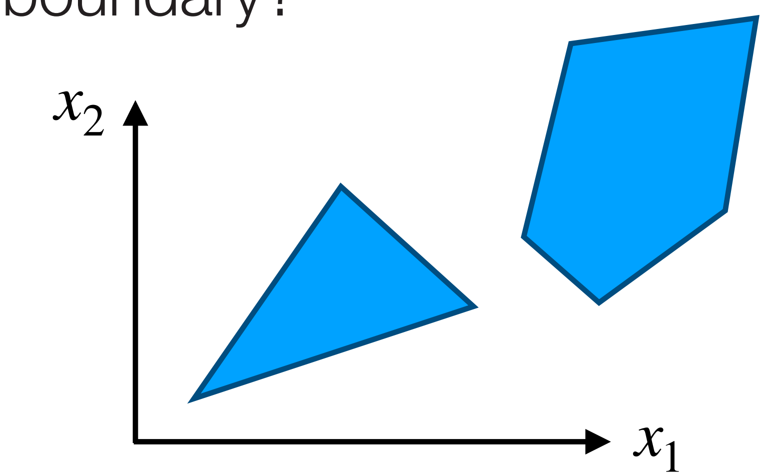


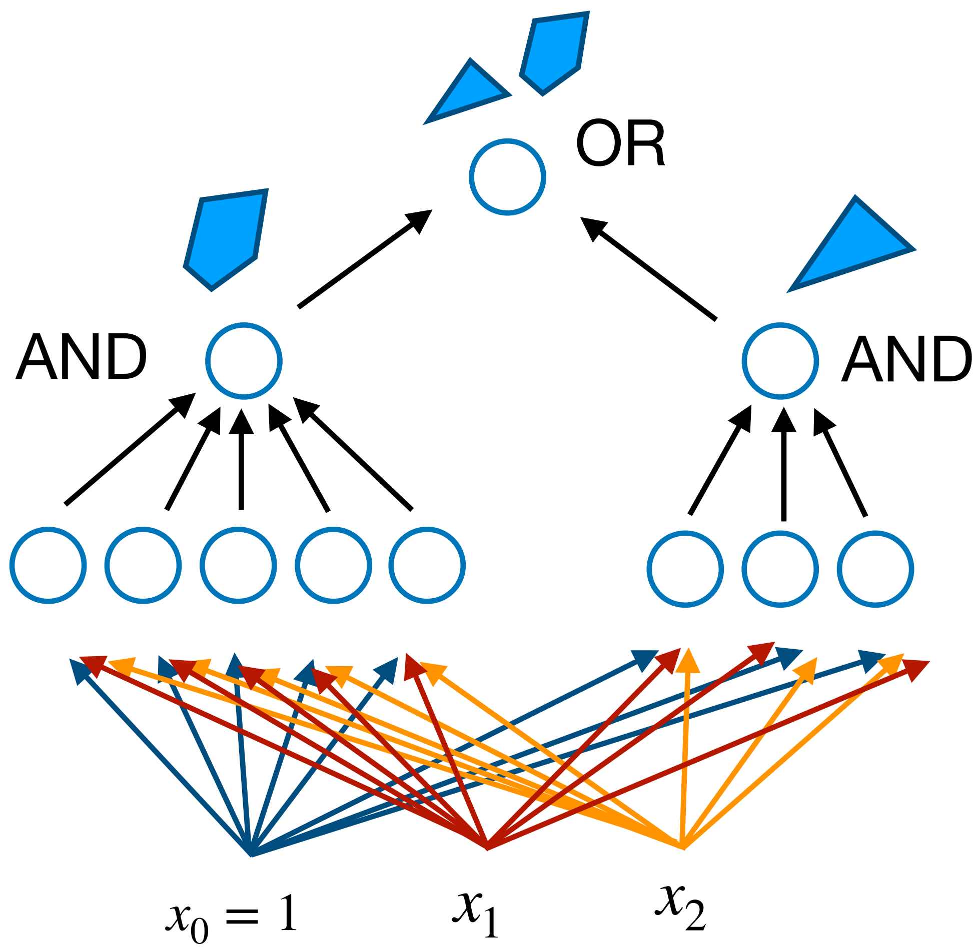
- 3 perceptrons: n_1, n_2, n_3 in a "hidden layer"
- each of these fires if (x_1, x_2) in light blue region
- output perceptron n_4 implements AND

➜ output neuron n_4 fires if in dark blue region

Neural Networks as Universal Approximators

- How to represent this complex, disconnected decision boundary?

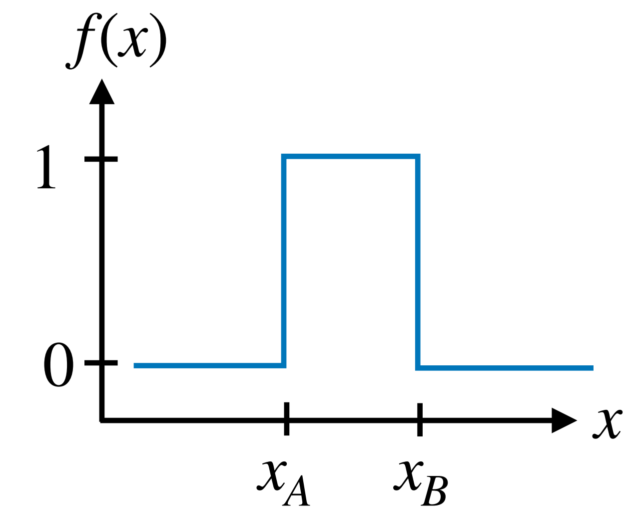




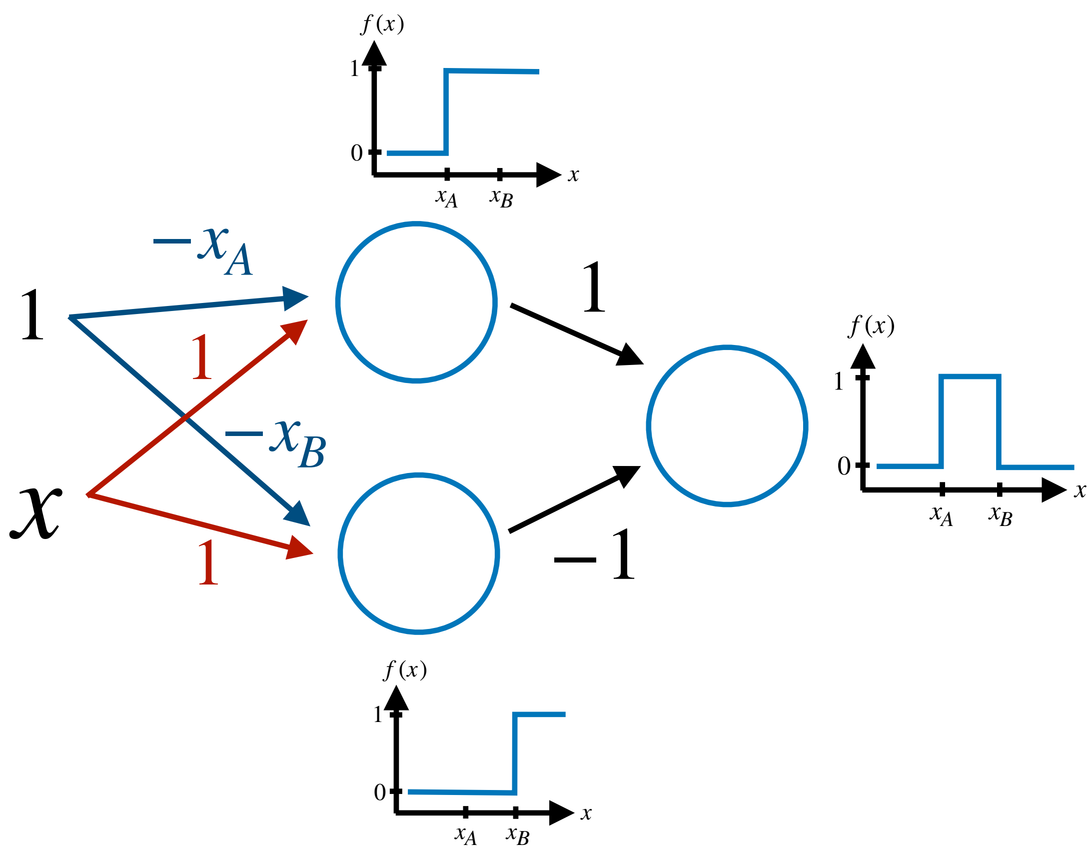
- here: 2 hidden layers
- can approximate arbitrary decision boundaries
- approximation can also be done with a single hidden layer, but may require many more neurons
- Motivation for **deep neural networks**: often fewer neurons needed

Neural Networks as Universal Approximators

- What about representing arbitrary functions?
  Yes, neural networks are *Universal function approximators*

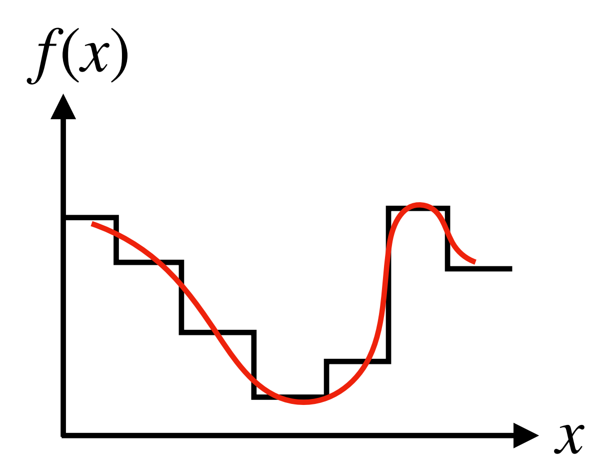


- For instance, how to represent a square pulse?



- output is 1 for $x_A < x \leq x_B$, 0 otherwise
- x_A, x_B can be arbitrarily chosen
- any (reasonable) function can be approximated with squares impulses

- can represent all functions from the domain of input values to the range of activation function
- can be extended to arbitrary dimensions

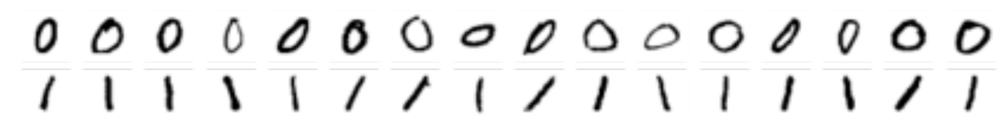


How to train your Neural Network

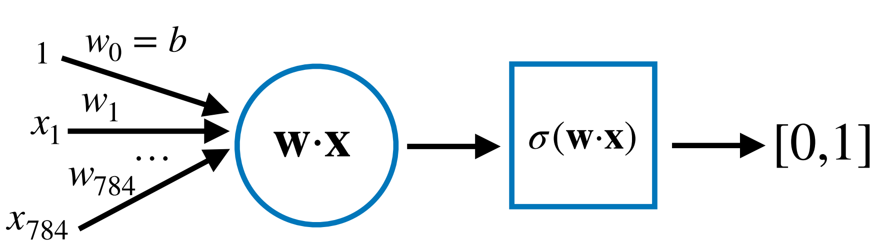
- Want to adjust weights (and biases) such that our network is “optimal” ➜ minimize **Cost function**
- Cost function depends on problem that the network is trying to solve

Example 1 (Classification, single neuron)

- Handwritten digit recognition
- Assume we only want to distinguish 0 and 1s



- Training data $D = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(n)}, y^{(n)})\}$, each $\mathbf{x}^{(i)}$ is a 28x28=784 pixel image, label $y^{(i)} \in \{0,1\}$
- Let us use “network” of single artificial neuron with sigmoid activation function (σ)



with $\hat{y}(\mathbf{w}, \mathbf{x}) = \sigma(\mathbf{w}\cdot\mathbf{x})$ is output of “network”

Note: $\hat{y} \in [0,1]$

binary cross-entropy

$$C(\mathbf{w}; D) = -\sum_{i=1}^{n} \left[y^{(i)} \ln \hat{y}(\mathbf{w}, \mathbf{x}^{(i)}) + (1 - y^{(i)}) \ln(1 - \hat{y}(\mathbf{w}, \mathbf{x}^{(i)})) \right]$$

- if $y = 1$, want $\hat{y} = 1$, ➜ $C = 0$
- if $y = 0$, want $\hat{y} = 0$, ➜ $C = 0$
- if $y = 1$ & $\hat{y} = 0$ or $y = 0$ & $\hat{y} = 1$ ➜ $C = \infty$

Example 1 (cont'd):

$$C(\mathbf{w}; D) = -\sum_{i=1}^{n} \left[y^{(i)} \ln \hat{y}(\mathbf{w}, \mathbf{x}^{(i)}) + (1 - y^{(i)}) \ln(1 - \hat{y}(\mathbf{w}, \mathbf{x}^{(i)})) \right]$$

- minimizing C w.r.t. weights $\mathbf{w}$ is a usual optimization problem
- e.g., may want to use (stochastic) gradient descent

$$\frac{\partial C(\mathbf{w}; D)}{\partial w_j} = -\sum_{i=1}^{n} x_j^{(i)} (y^{(i)} - \hat{y}(\mathbf{w}, \mathbf{x}^{(i)}))$$

Homework: show this!

$$\nabla_{\mathbf{w}} C(\mathbf{w}; D) = -\sum_{i=1}^{n} \mathbf{x}^{(i)} (y^{(i)} - \hat{y}(\mathbf{w}, \mathbf{x}^{(i)}))$$

vector form

Aside:

- This is quite similar to logistic regression
- Recall in log. regression: Model the log-odds as a linear function of the predictor values

$$\ln\left(\frac{p}{1-p}\right) \sim \boldsymbol{\beta} \cdot \mathbf{x} \qquad \begin{aligned} p &= \Pr(y = 1 \,|\, \mathbf{x}) \\ 1 - p &= \Pr(y = 0 \,|\, \mathbf{x}) \end{aligned} \qquad \boldsymbol{\beta} \cdot \mathbf{x} = \beta_0 1 + \beta_1 x_1 + \ldots + \beta_n x_n$$

➜ Probabilistic classification: $\Pr(y = 1 \,|\, \mathbf{x}) = \dfrac{1}{1 + e^{-\boldsymbol{\beta}\mathbf{x}}} = \sigma(\boldsymbol{\beta} \cdot \mathbf{x})$

Example 2 (Regression, network):

- Training data $D = \{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), \ldots, (\mathbf{x}^{(n)}, \mathbf{y}^{(n)})\}$, $\mathbf{x}^{(i)} \in R^p$
  - $y^{(n)} \in R^q$: regression problem

- Cost function for single data point

$$C(\mathbf{w}; \mathbf{x}^{(i)}, \mathbf{y}^{(i)}) = \frac{1}{2} \|\mathbf{y} - \hat{\mathbf{y}}(\mathbf{w}; \mathbf{x}^{(i)})\|^2$$

"squared error loss"

- Cost function for n data points (MSE)

$$C(\mathbf{w}; D) = \frac{1}{n} \sum_{i=1}^{n} C(\mathbf{w}; \mathbf{x}^{(i)}, \mathbf{y}^{(i)})$$

- Gradient

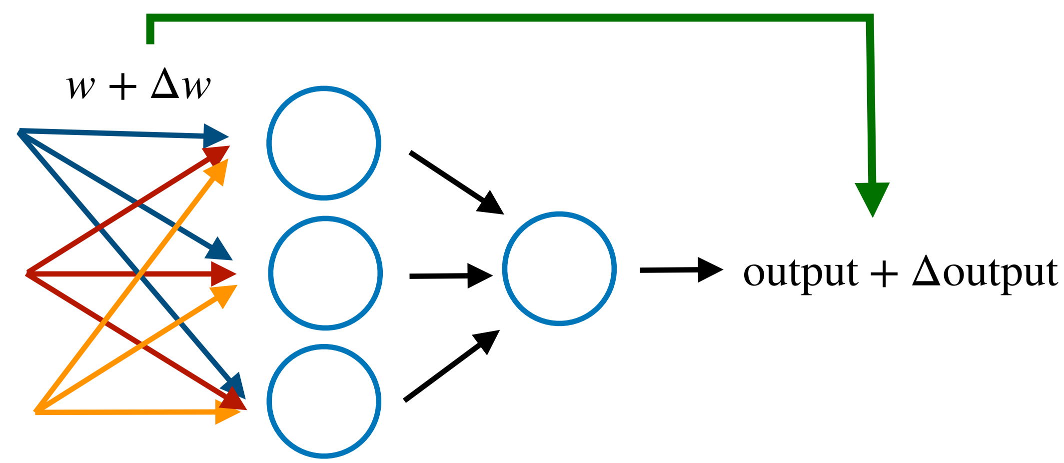
$$\frac{\partial C(\mathbf{w}; D)}{\partial w_k} = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial C(\mathbf{w}; \mathbf{x}^{(i)}, \mathbf{y}^{(i)})}{\partial w_k}$$

- Update weights

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{\partial C(\mathbf{w}; D)}{\partial w_k}$$

η is learning rate

- Computational Challenge: Network has millions of weights, often large data sets
- How to calculate gradient $\dfrac{\partial C(\mathbf{w}; \mathbf{x}^{(i)}, \mathbf{y}^{(i)})}{\partial w_k}$ efficiently?



Backpropagation - Idea

- forward pass to calculate output of all neurons
- calculate output error
- propagate this error backwards through the network to calculate the gradient w.r.t. all weights via the chain rule
- Then we can use standard methods to update weights (e.g., stochastic gradient descent)