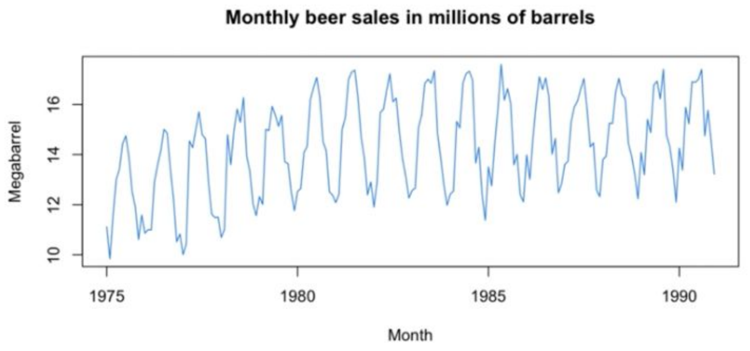
Time series Analysis

What is a time series

- Complex topic :
    - information theory
    - signal
    - probability/statistics
    - artificial intelligence/machine learning
    - optimization
    - Computer science (python)
    - Financial mathematics
- http://www.laurentoudre.fr/ast.html
- http://www.laurentoudre.fr/signalml.html
- https://scikit-learn.org/stable/
- https://github.com/fastai/course22
- https://github.com/fastai/fastbook
- https://github.com/sduprey/initiation_python_finance
- https://github.com/sduprey/timeseries_ressources
- Continuous value (no sequence, discrete time data)
- Multivariate vector time series (brain signals ECG, multiple stocks)



Modeling vs Predicting : narrowing down the determinism
A framework which encompasses AI and econometry

$$y_t = f(y_{t-1}, \ldots, y_{t-p}; \theta) + \varepsilon_t$$

- AI predilection domain: complex function with embeddings, features engineering and a small pure random part (see for example the Rossman sales kaggle competition)
- Econometrics predilection domain : function very simple (just a scalar for a random walk) (try for example to predict the bitcoin price)

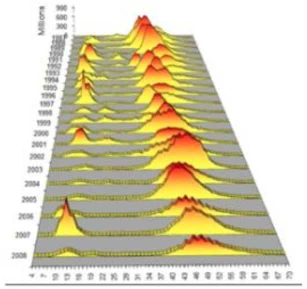
Modeling vs Predicting : narrowing down the determinism
A framework which encompasses AI and econometry

$$y_t = f(y_{t-1}, \dots, y_{t-p}; \theta) + \varepsilon_t$$

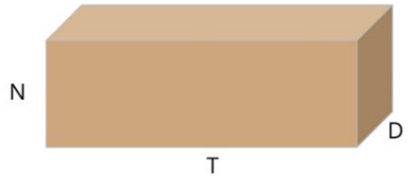
- Gives a functional form to your time series
- Gives you insight into the behavior of the time series
- "Why is the time series mean-reverting?"
- "Why does the time series grow unbounded?"
- "Is the time series predictable?"
- Someone who doesn't know how to model might waste their time trying to predict something which is unpredictable! (e.g. a coin flip)

Dataframe (R, Pandas, PyTorch/TensorFlow tensors)

	New York City	London	Tokyo	Paris
1990-01-01	1	2	3	4
1990-01-02	5	6	7	8
1990-01-03	9	10	11	12
1990-01-04	13	14	15	16



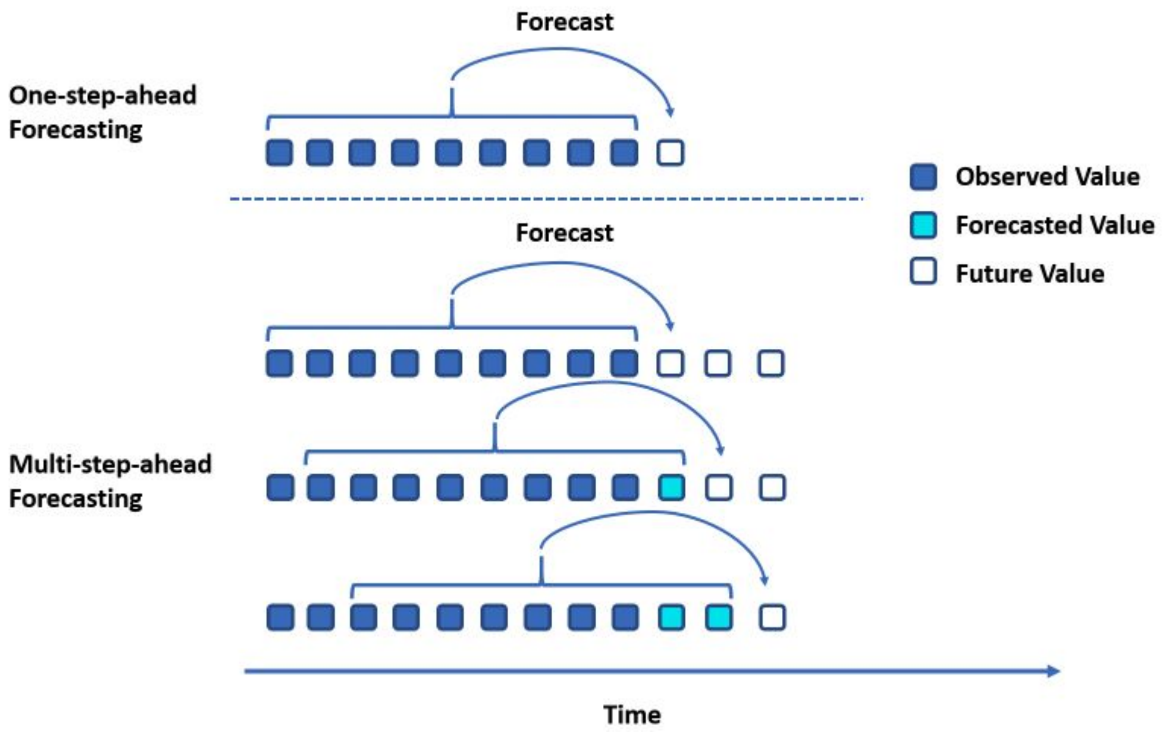
- N x T x D (a box in 3-D space)
- Automatically think of this, whenever you see / hear "N x T x D"
- It really helps, and makes it less abstract



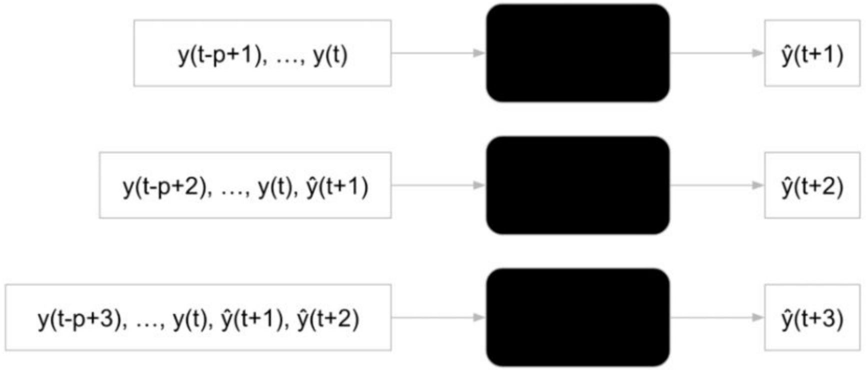
Generate random tensors/time series

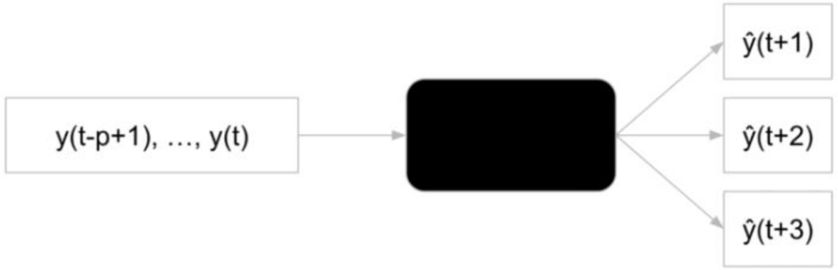
- np.random.randn(3,3,3)
- Generate from different probability distributions
- Distribution fitting with scikit-learn

1-step forecast versus multi-steps forecast

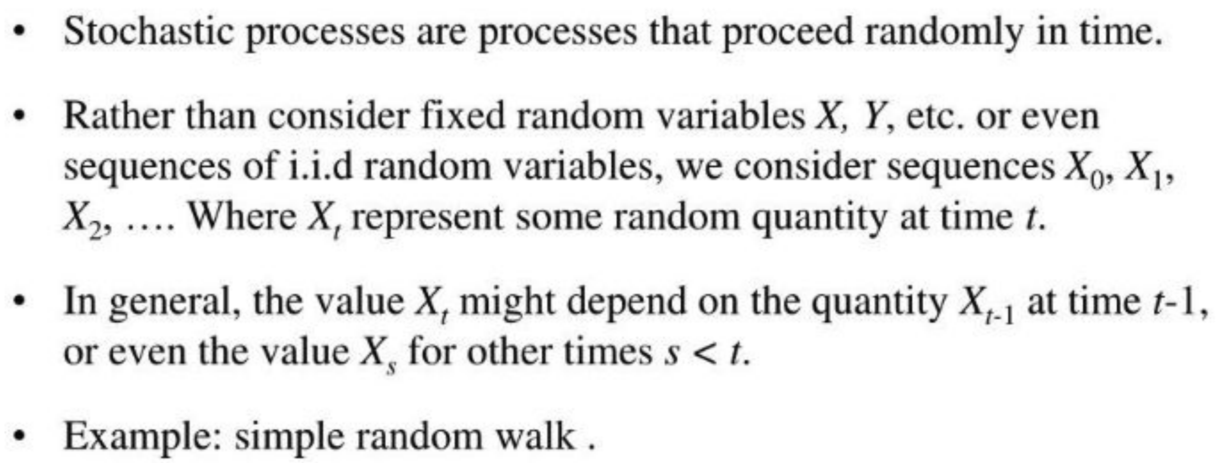


Incremental multi-steps forward versus multi-output





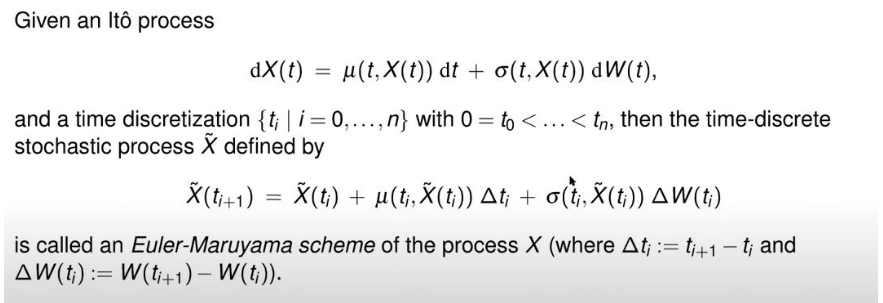
Stochastic processes in a nutshell



- Stochastic processes are processes that proceed randomly in time.
- Rather than consider fixed random variables X, Y, etc. or even sequences of i.i.d random variables, we consider sequences X_0, X_1, X_2, …. Where X_t represent some random quantity at time t.
- In general, the value X_t might depend on the quantity $X_{t\text{-}1}$ at time t-1, or even the value X_s for other times $s < t$.
- Example: simple random walk .

Going more into maths:

- **Markov process**
- **Martingales**
- **Discretization of continuous processes**



Given an Itô process

$$\mathrm{d}X(t) = \mu(t, X(t))\,\mathrm{d}t + \sigma(t, X(t))\,\mathrm{d}W(t),$$

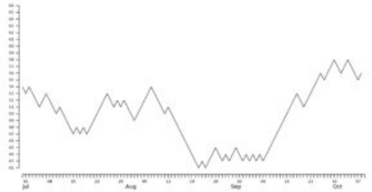
and a time discretization $\{t_i \mid i = 0, \ldots, n\}$ with $0 = t_0 < \ldots < t_n$, then the time-discrete stochastic process $\tilde{X}$ defined by

$$\tilde{X}(t_{i+1}) = \tilde{X}(t_i) + \mu(t_i, \tilde{X}(t_i))\,\Delta t_i + \sigma(t_i, \tilde{X}(t_i))\,\Delta W(t_i)$$

is called an *Euler-Maruyama scheme* of the process X (where $\Delta t_i := t_{i+1} - t_i$ and $\Delta W(t_i) := W(t_{i+1}) - W(t_i)$).

Random walk for instance

Discrete random walk



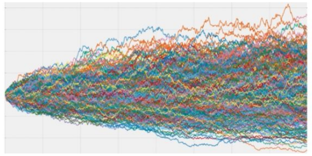
$p_0 = \text{some initial value}$

$p_1 = p_0 + e_1,\ \text{where } e_1 \in \{-1, +1\}$

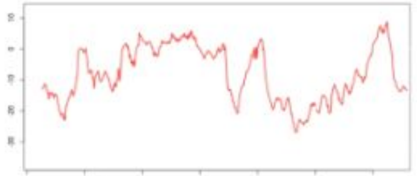
$p_2 = p_1 + e_2$

...

- Imagine yourself walking - you take one step **left** or **right** based on a coin flip - that's this random walk!
- Can't predict the future (50% chance of being correct)



Gaussian random walk

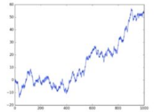


$p_0 = \text{some initial value}$

$p_1 = p_0 + e_1,\ \text{where } e_i \sim \mathcal{N}(0, \sigma^2)$

$p_2 = p_1 + e_2$

...



Log Prices

- Consider a random walk with drift

$$p_t = p_{t-1} + \mu + e_t,\ e_t \sim \mathcal{N}(0, \sigma^2)$$

- Take p(t-1) to the LHS - this is now the log return

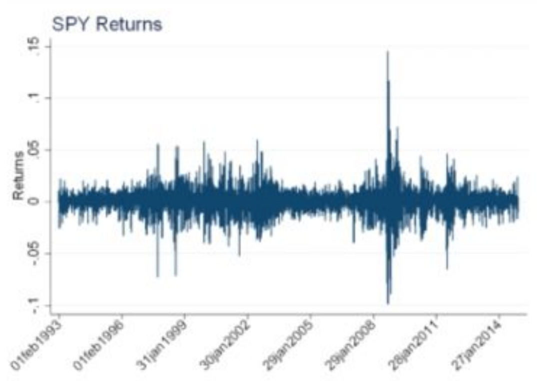
$$r_t = p_t - p_{t-1} = \mu + e_t$$

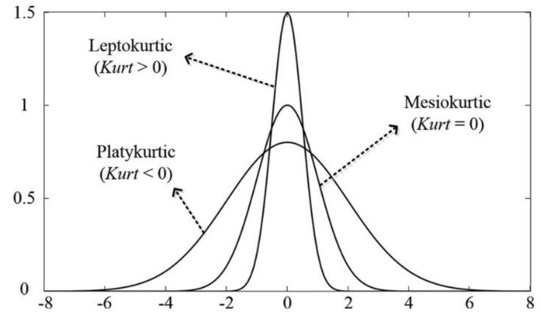
- The log return is therefore distributed as follows

$$r_t \sim \mathcal{N}(\mu, \sigma^2)$$

Stylized facts about financial time-series which invalidate the random walk hypothesis

- Volatility clustering
- Shock asymmetry
- Leptokurtic residuals





Leptokurtic (or platykurtic) degree difference of curves with different kurtosis.

Markov property

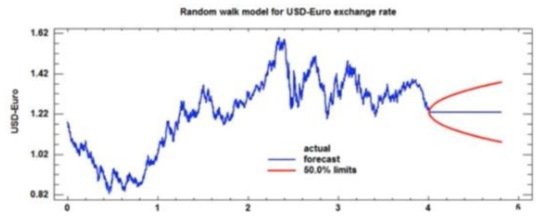
$$p(w_t \mid w_{t-1}, w_{t-2}, \ldots, w_0) = p(w_t \mid w_{t-1})$$

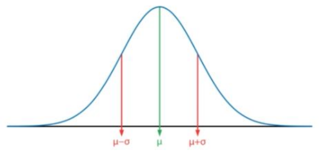
Gaussian random walk

$$x(t) = x(t-1) + e(t),\; e(t) \sim \mathcal{N}(0, \sigma^2)$$
$$x(t) \sim \mathcal{N}(x(t-1),\, \sigma^2)$$

$$var\{\, x(t+\tau)\,\} \;=\; ?$$
$$x(t+1) = x(t) + e(t+1)$$
$$x(t+2) = x(t+1) + e(t+2) = x(t) + e(t+1) + e(t+2)$$
$$\ldots$$
$$x(t+\tau) = x(t) + e(t+1) + \ldots + e(t+\tau)$$

Square root of time growing confidence interval
Central limit theorem :
Converges to a gaussian

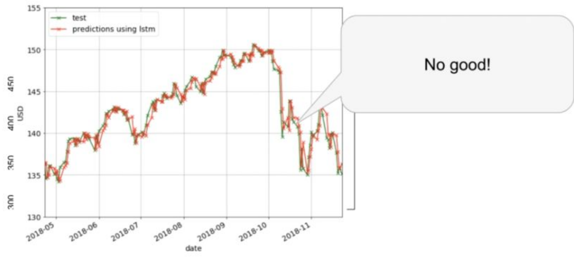


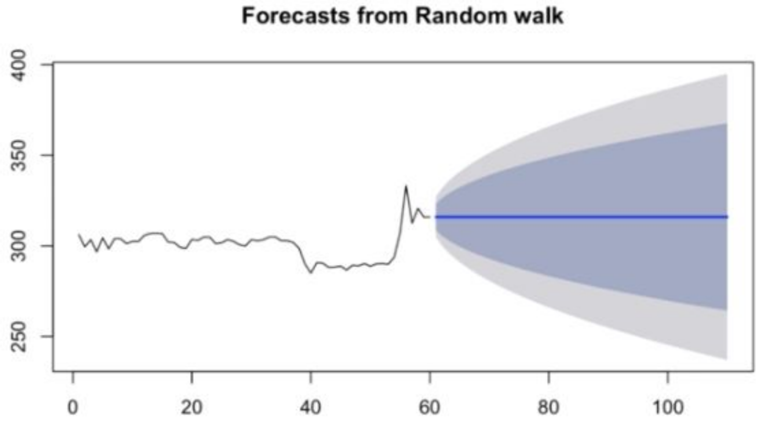


$$If\ var\{e(t)\} = \sigma^2,\ then\ var \sum_{k=1}^{\tau} e(t+k) = \tau\sigma^2,\ or\ sd\{x(t+\tau)\} = \sqrt{\tau}\,\sigma$$

Naive forecast or the importance of a baseline

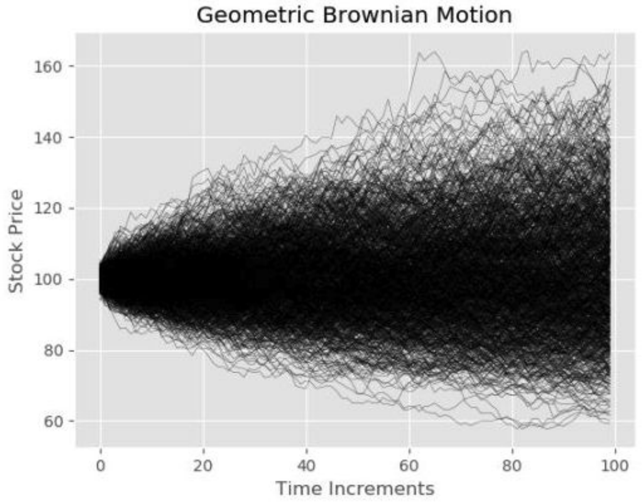
- Compare your results against existing state of the art
- Be careful :
    - the naïve forecast looks good for martingale processes
    - In-sample versus out-of-sample forecasts
    - You can beat the naive forecast in-sample but not out-sample
- **The naive forecast is the best for a random walk (for any martingales)**





Option pricing through Monte-Carlo simulation

- https://github.com/sduprey/timeseries_ressources/Black_Scholes_option_pricing_through_Monte_Carlo_simulation.ipynb



$$dS = S\mu dt + S\sigma dW(t)$$

$$dG = (\frac{\partial G}{\partial S} S\mu + \frac{\partial G}{\partial t} + \frac{1}{2}\frac{\partial^2 G}{\partial S^2} S^2\sigma^2)dt + \frac{\partial G}{\partial S} S\sigma dW(t)$$

$$\frac{\partial G}{\partial S} = \frac{1}{S} \ , \ \frac{\partial G}{\partial t} = 0 \ , \ \frac{\partial^2 G}{\partial S^2} = -\frac{1}{S^2}$$

$$\begin{aligned} dG &= (\frac{1}{S}S\mu + 0 - \frac{1}{2}\frac{1}{S^2}S^2\sigma^2)dt + \frac{1}{S}S\sigma dW(t) \\ &= (\mu - \frac{\sigma^2}{2})dt + \sigma dW(t) \end{aligned}$$

$$lookback = e^{-rT}[\frac{1}{N}\sum_{i=1}^{N}(S_{max} - K)^+]$$

Forecasting metrics (like regression)

- Sum of squared errors (max likelihood when the errors are normally distributed) $E = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$

- Mean squared error $E = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \approx E\left((y - \hat{y})^2\right)$

- Root mean squared error (same unit) $E = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2}$

- Mean absolute error $E = \frac{1}{N} \sum_{i=1}^{N} |y_i - \hat{y}_i|$

R squared

- Not an error - we want it to be bigger, not smaller
- If your model makes perfect predictions, then MSE=0, R^2=1
- R^2=0 means your model does no better than predicting $\bar{y}$

$$E = 1 - \frac{SSE}{SST} = 1 - \frac{MSE}{Var(Y)}$$

$$where\ SST = \sum_{i=1}^{N}(y_i - \bar{y})^2,\ Var(Y) = \frac{1}{N}SST$$

Scikit-learn

- For classification models: model.score(X, Y) returns accuracy
- For regression models: model.score(X, Y) returns R^2

Problems of relativity

- Accuracy is not the best metrics for an imbalanced classifier

$$E = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{y_i - \hat{y}_i}{y_i}\right|$$

- MAPE (mean absolute percentage error)

- sMAPE

$$E = \frac{1}{N}\sum_{i=1}^{N}\frac{|y_i - \hat{y}_i|}{(|y_i| + |\hat{y}_i|)\,/\,2}$$

Common transformations : analog to features engineering in artificial intelligence

- Power transform $$y'(t) = y(t)^{\gamma}$$

- Log transform $$y'(t) = \log y(t) \ or \ \log(y(t) + 1)$$

- Box-Cox transform
$$y'(t) = \frac{y(t)^{\lambda} - 1}{\lambda} \quad if\ \lambda \neq 0$$
$$y'(t) = \log y(t) \quad if\ \lambda = 0$$

Since:
$$\lim_{\lambda \to 0} \frac{x^{\lambda}-1}{\lambda} = \ln x$$

Why the log transform is fundamental

- Watch fast.ai on random forest : log uniformises large tail distributions
- In more standard econometrics, it is all about stationarity
- Rescaling data to more linear trends (decibels are the best example)
- It is part of a broader trend : features engineering for tabular data

https://github.com/fastai/fastbook/blob/master/09_tabular.ipynb

SMA

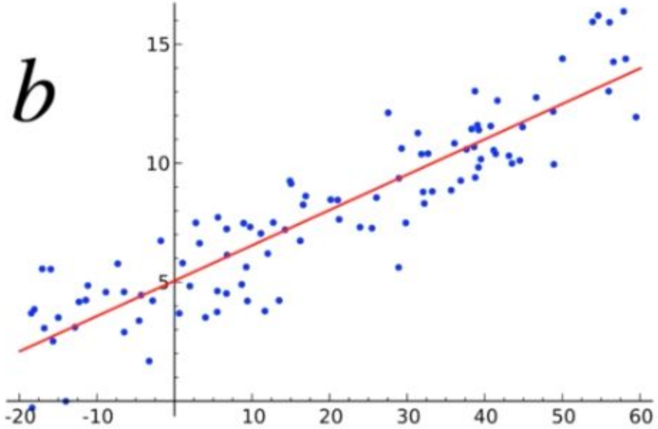
ARIMA vs Exponential smoothing

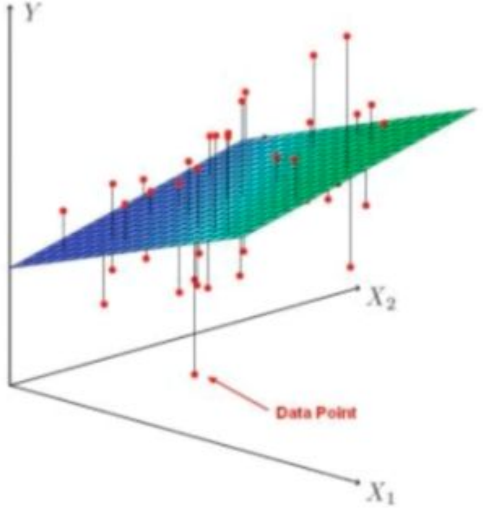
- Exponential smoothing is very specific (linear trends, seasonality)
- ARIMA imposes no such structure
- It is more “machine learning”-like

Autoregressive models

$$\hat{y} = w_1 x_1 + w_2 x_2 + b$$

$$\hat{y} = mx + b$$





AR(p) : auto-regressive processus

$$\hat{y}_t = b + \varphi_1 y_{t-1} + \varphi_2 y_{t-2} + \ldots + \varphi_p y_{t-p}$$

- Suppose our data is: $y_1, y_2, \ldots, y_{10}$
- In ML, we say X has shape N x D, but for ARIMA we'll stick with D == p

y1	y2	y3
y2	y3	y4
y3	y4	y5
y4	y5	y6
y5	y6	y7
y6	y7	y8
y7	y8	y9

This is our “X”

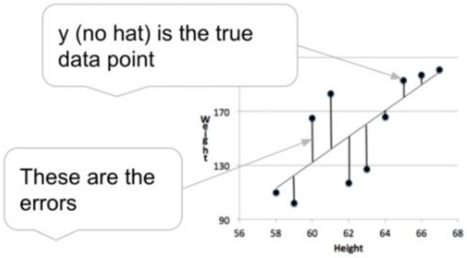
y4
y5
y6
y7
y8
y9
y10

This is our “Y”

$$y_t = b + \varphi_1 y_{t-1} + \varphi_2 y_{t-2} + \ldots + \varphi_p y_{t-p} + \varepsilon_t$$

$$\varepsilon_t \sim \aleph(0, \sigma^2)$$

$$\hat{y}_t = E(y_t)$$

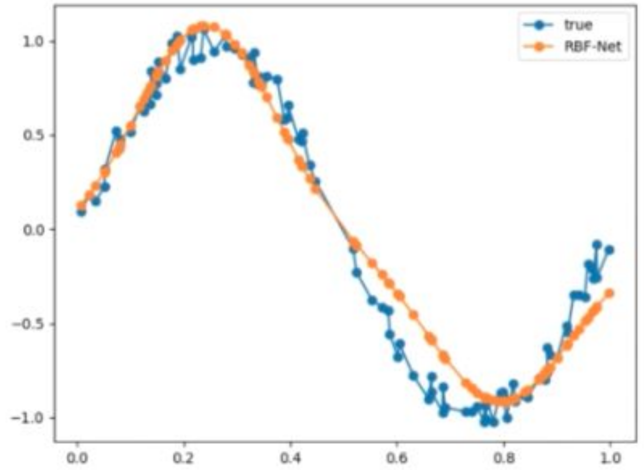


Machine Learning is the next step

- Linear models aren't that powerful (only lines or planes)
- Why stick to linear regression?
- ARIMA helps us understand the modeling and statistical properties

```
model = NeuralNetwork()
model.fit(X, Y)

model = RandomForest()
model.fit(X, Y)
```



MA(q)

$$y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \dots + \theta_q \varepsilon_{t-q}$$

Moving around the average c:

Zero!

$$E(y_t) = E(c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \dots + \theta_q \varepsilon_{t-q})$$

$$= c$$

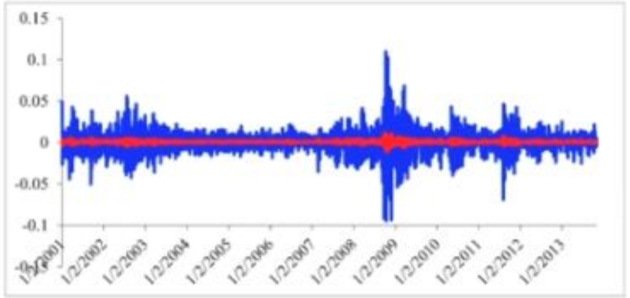
ARMA(p,q)

- ARMA(p, q) = AR(p) + MA(q)

$$y_t = b + \boxed{\varphi_1 y_{t-1} + \dots + \varphi_p y_{t-p}} + \boxed{\theta_1 \varepsilon_{t-1} + \dots + \theta_q \varepsilon_{t-q}} + \varepsilon_t$$

Differencing : detrending

- Where have we seen this?
  - Log returns - the difference of log prices: $r_t = p_t - p_{t-1}$
  - Holt's Linear Trend Model: $b_t = \beta(l_t - l_{t-1}) + (1-\beta)b_{t-1}$

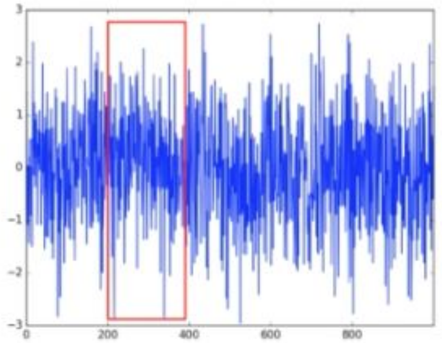


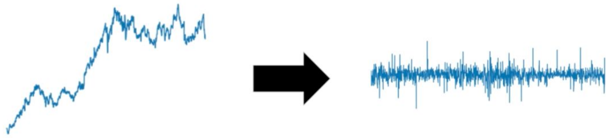
$$Given:\ \{y_t\} = \{y_1, y_2, ..., y_T\}\ (some\ time\ series)$$

$$Differenced\ Series:\ \Delta y_t = y_t - y_{t-1}$$

ARMA modelling needs stationary time series

- When fitting an ARMA model, we want the data to be close to stationary
- Stationary = Does not change over time
- Stationarity is nice: mean, variance, autocorrelation, ... will be constant over time
  - Recall: linear models fit well when there is strong correlation between inputs / output
- Each "window" of the time series is like a "training point" for fitting the model





I(d) and ARIMA(p,d,q)

- An I(d) process is a process that is stationary after differencing d times
- We say it's integrated to order d
- ARIMA(p, d, q) is just a model where we've differenced d times before applying ARMA(p, q)

- ARIMA(p, 0, 0) is AR(p)
- It's also ARMA(p, 0)
- ARIMA(0, 0, q) is ARMA(0, q) and MA(q)
- ARIMA(0, d, 0) is I(d)

Random Walk

- ARIMA(0, 1, 0) is I(1) and this is a random walk

Differenced time series

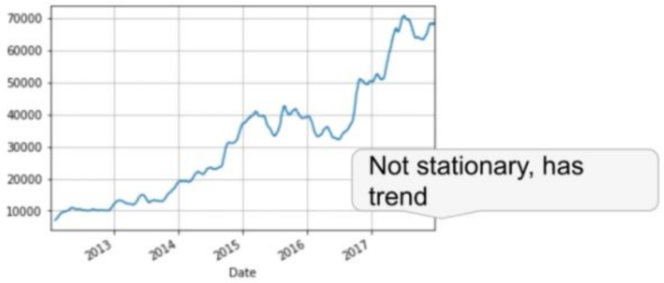
Absence of AR and MA components

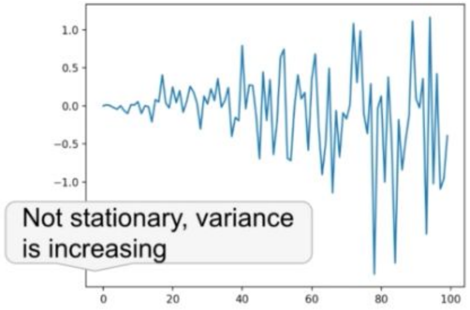
$$\Delta y_t = \varepsilon_t$$

$$y_t - y_{t-1} = \varepsilon_t$$

$$y_t = y_{t-1} + \varepsilon_t$$

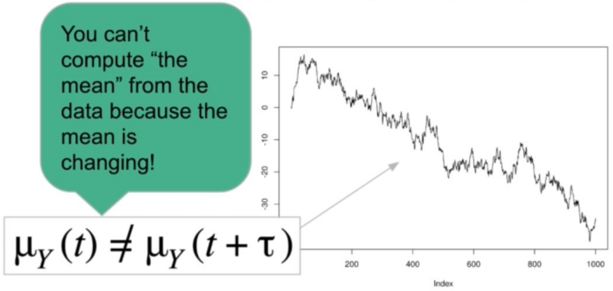
Stationarity



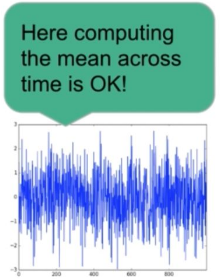


- Loosely, the distribution of the random variables in the time series does not change over time
- E.g. mean and variance will always be the same

- If a time series is nonstationarity, then you might need different models at different points in time!
- For nonstationary time series, you can't treat data points at different times like "samples" (e.g. computing the mean, variance)

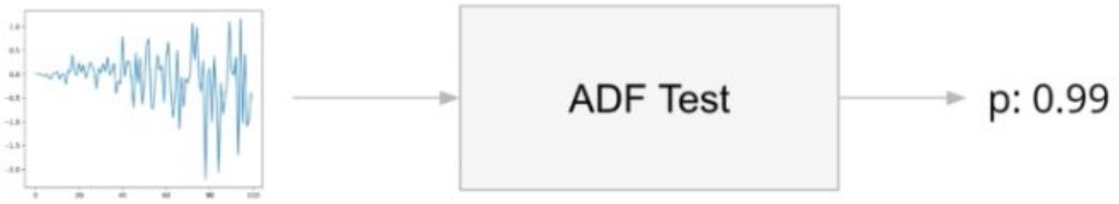


$$\mu_Y(t) \neq \mu_Y(t+\tau)$$

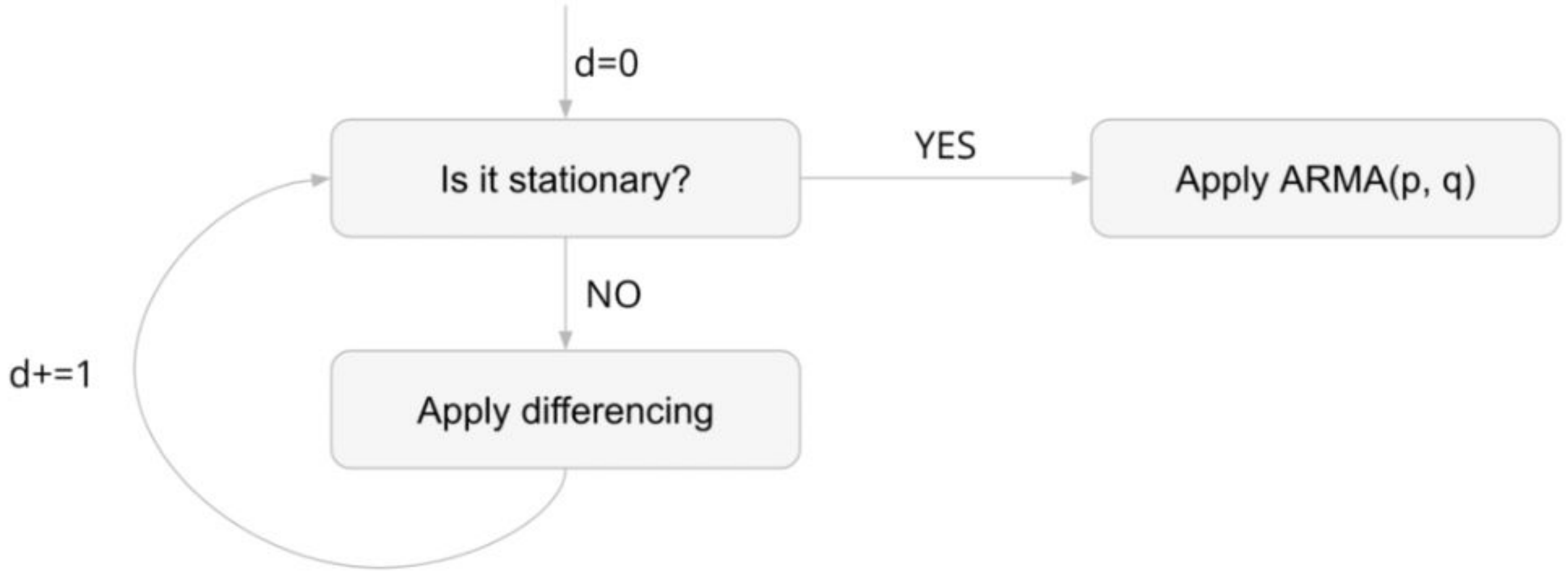


Testing for stationarity

- We use the Augmented Dickey-Fuller Test (ADF Test)
- Think of it like an API:
  - Given: null hypothesis, alternative hypothesis
  - Input: time series, Output: p-value
  - Action: accept or reject the null hypothesis
- For ADF test:
  - Null: time series is non-stationary
  - Alternative: time series is stationary



How to use ADF test in selecting d in ARIMA



Strong vs Weak

- Strong: the entire distribution does not change over time

$$F_Y(y_{t_1+\tau}, y_{t_2+\tau}, \ldots, y_{t_n+\tau}) = F_Y(y_{t_1}, y_{t_2}, \ldots, y_{t_n}),\ \forall \tau, t_1, t_2, \ldots, t_n$$

- Weak : First order (mean) and second-order statistics (covariance) stay the same
  - The mean does not change over time

$$\mu_Y(t) = \mu_Y(t+\tau)\ for\ all\ \tau$$

  - The autocovariance does not change over time

$$K_{YY}(t_1, t_2) = K_{YY}(t_1 - t_2, 0)\ for\ all\ t_1, t_2$$

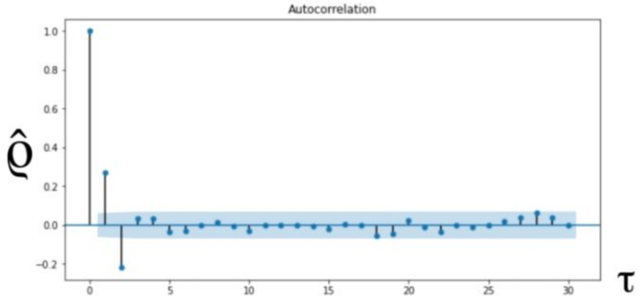
$$Autocovariance: \; cov(Y_{t_1}, Y_{t_2})$$

Autocorrelation function

$$Autocorrelation: \; \frac{cov(Y_{t_1}, Y_{t_2})}{\sigma_Y(t_1)\sigma_Y(t_2)}$$

$$Stationary: \; \varrho(Y(t_1), Y(t_2)) = \varrho(t_1 - t_2) = \varrho(\tau)$$

- Also known as correlogram
- Autocorrelation is to autocovariance as correlation is to covariance
- Auto = Self (both RVs come from the same time series)



How to determine q in MA(q)

- Assign q to be the maximum non-zero lag
- E.g. in below chart, q = 2
- Usually, the ACF for lags < q are also non-zero

Why does it work ?

- This can be derived mathematically! (we won't do it right now)
- For MA(1):

$$y_t = c + \theta_1 \varepsilon_{t-1} + \varepsilon_t,\ where\ \varepsilon_t \sim \aleph(0, \sigma^2)$$

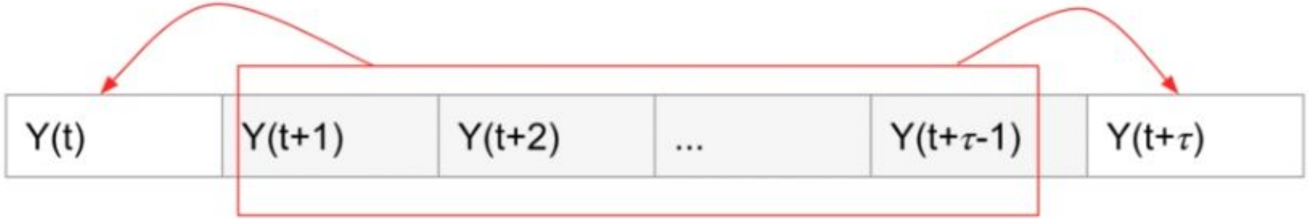
$$\varrho(1) = \frac{\theta_1}{1+\theta_1{}^2},\ \varrho(\tau) = 0\ for\ \tau > 1$$

- For MA(2):

$$\varrho(1) = \frac{\theta_1+\theta_1\theta_2}{1+\theta_1{}^2+\theta_2{}^2},\ \varrho(2) = \frac{\theta_2}{1+\theta_1{}^2+\theta_2{}^2},\ \varrho(\tau) = 0\ for\ \tau > 2$$

PACF of order p of AR(p)

- Definition: The PACF at lag τ is the autocorrelation between Y(t) and Y(t+τ), *conditioned on* Y(t+1), Y(t+2), ..., Y(t+τ-1)



$$\varphi(\tau,\tau) = corr(Y_{t+\tau} - \hat{Y}_{t+\tau}, Y_t - \hat{Y}_t)$$

$$\hat{Y}_{t+\tau} = \beta_0 + \beta_1 Y_{t+1} + \beta_2 Y_{t+2} + \ldots \beta_{\tau-1} Y_{t+\tau-1}$$

$$\hat{Y}_t = \beta_0' + \beta_1' Y_{t+1} + \beta_2' Y_{t+2} + \ldots \beta_{\tau-1}' Y_{t+\tau-1}$$

GARCH theory

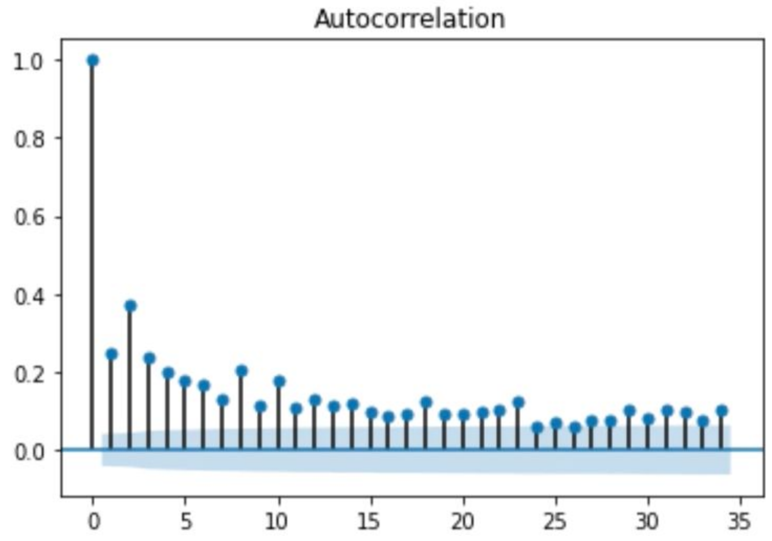
ARIMA: used to model time series mean (variance = 0)

$$y_t = f(y_{t-1}, \ldots, y_{t-p}; \theta) + \varepsilon_t$$

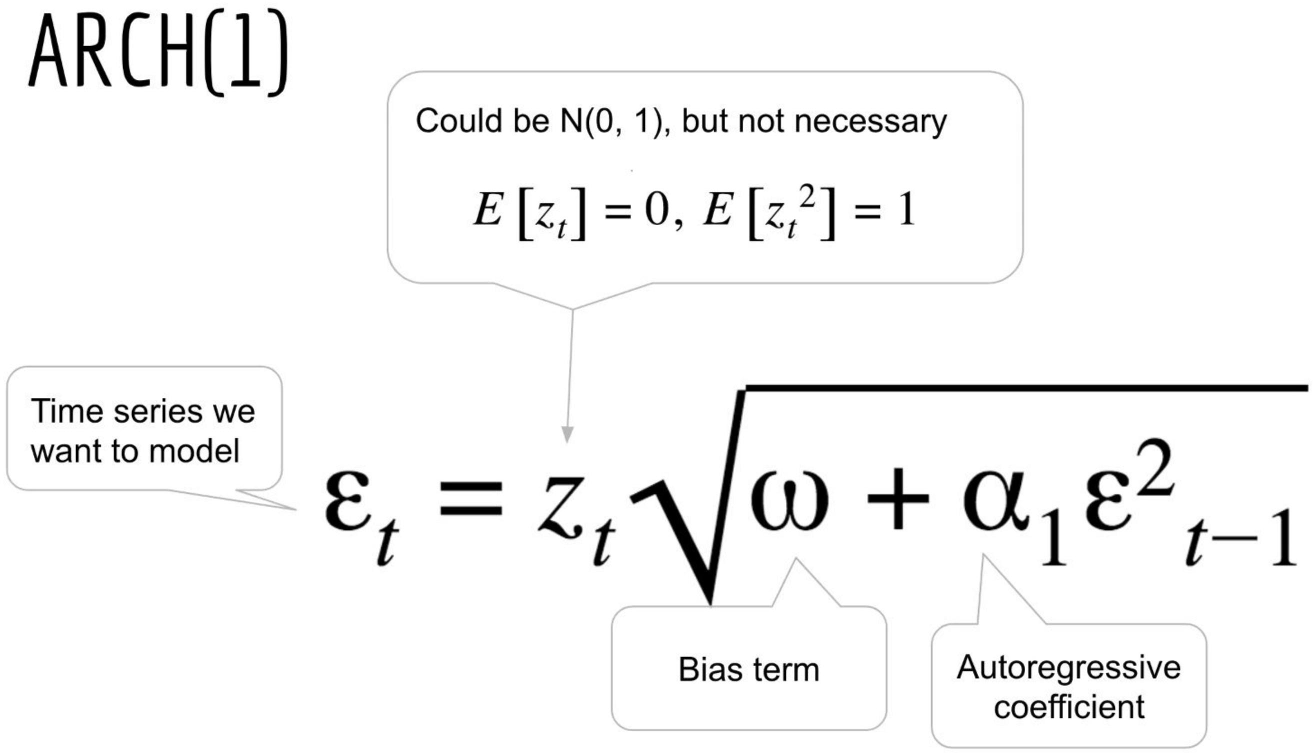
GARCH: used to model time series variance (mean = 0)

ACF of squared returns show autoregressivity in variance

- For stocks, log returns ACF shows randomness (I(1) is the best model)
- The ACF of squared log returns does not look random at all



ARCH(1)


ARCH(1)
Could be N(0, 1), but not necessary
Time series we want to model
Bias term
Autoregressive coefficient

$$E\left[z_t\right] = 0,\ E\left[{z_t}^2\right] = 1$$

$$\varepsilon_t = z_t \sqrt{\omega + \alpha_1 \varepsilon^2_{t-1}}$$

$\varepsilon_t = z_t \sigma_t$ => epsilon has variance sigma squared

$$\varepsilon_t{}^2 = z_t{}^2(\omega + \alpha_1 \varepsilon^2{}_{t-1})$$

$$\varepsilon_t = z_t \sigma_t$$

$$\frac{\varepsilon_t{}^2}{z_t{}^2} = \boxed{\sigma_t{}^2 = \omega + \alpha_1 \varepsilon^2{}_{t-1}}$$

- There must be constraints, since variance cannot be negative

$$\varepsilon_t = z_t\sqrt{\omega + \alpha_1 \varepsilon^2{}_{t-1}}$$

- General constraints

$$\omega > 0,\ 1 \geq \alpha_1 \geq 0$$

- Constraint for stationarity and finite variance

$$\alpha_1 < 1$$

$$E\left[\varepsilon_t^{\,2} \mid \varepsilon_{t-1}, \varepsilon_{t-2}, \ldots, \varepsilon_{t-p}\right] = \sigma_t^{\,2}$$

$$E\left[\varepsilon_t^{\,2}\right] = \ ?\ (\textit{call it}\ \sigma^2)$$

$$E\left[\varepsilon_t^{\,2}\right] = E\left[z_t^{\,2}\right] E\left[\omega + \alpha_1 \varepsilon_{t-1}^{\,2}\right]$$

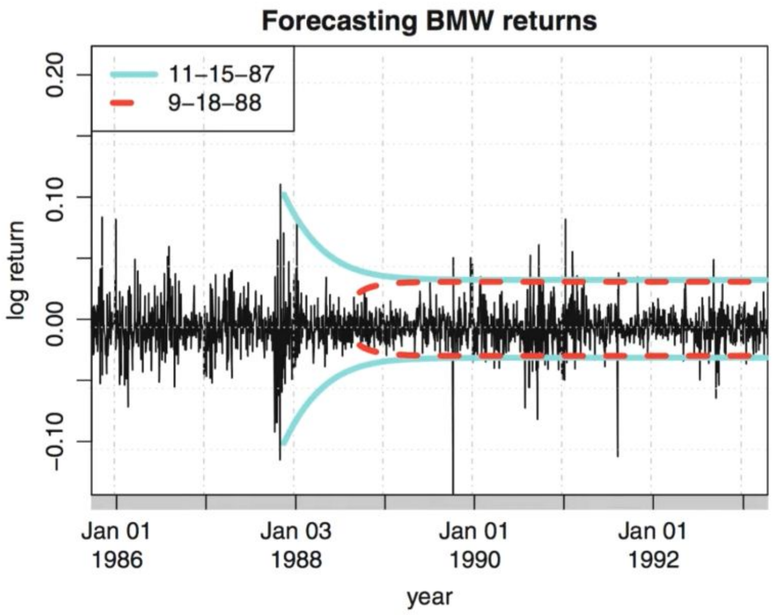
$$\sigma^2 = 1 \times (\omega + \alpha_1 \sigma^2)$$

$$(1 - \alpha_1)\sigma^2 = \omega$$

$$\sigma^2 = \frac{\omega}{1 - \alpha_1}$$

Long term forecast converge to the unconditional variance

- Our forecast of variance will also converge to unconditional variance
- Meaning: ARCH and GARCH are useful for short-term forecasts
- Makes sense: we cannot predict Elon Musk making a tweet about Bitcoin, or some country banning a cryptocurrency exchange
- Obvious example: COVID-19
- Imagine if time series models could predict world-scale events! (of course, this is unreasonable)
- Aside from insider trading, you probably cannot predict these



ARCH(p)

$$\varepsilon_t = z_t \sqrt{\omega + \sum_{\tau=1}^{p} \alpha_\tau \varepsilon^2{}_{t-\tau}}$$

$$E\left[\varepsilon_t \mid \varepsilon_{t-1}, \varepsilon_{t-2}, \ldots, \varepsilon_{t-p}\right] = 0$$

$$E\left[\varepsilon_t\right] = 0$$

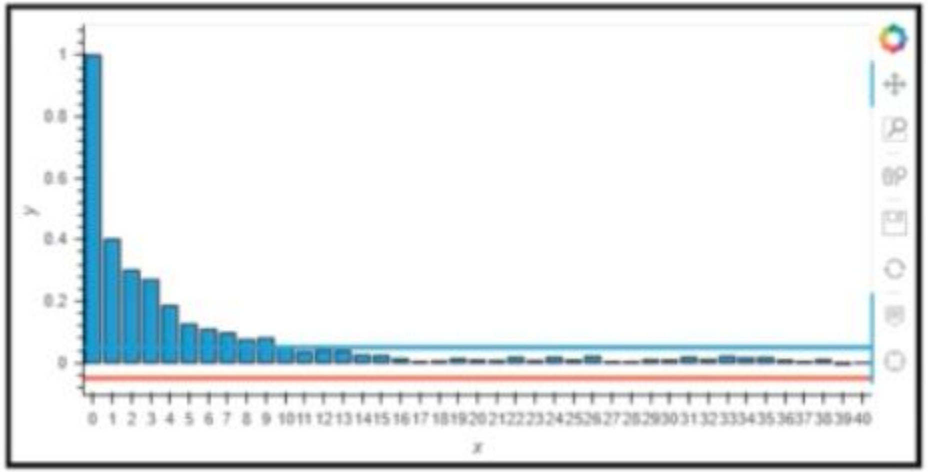
$$cov(\varepsilon_t, \varepsilon_{t-s}) = 0, \ \forall s \neq 0$$

$$\varepsilon_t \perp \varepsilon_{t-s} \quad \rightarrow \quad cov(\varepsilon_t, \varepsilon_{t-s}) = 0$$

$$cov(\varepsilon_t, \varepsilon_{t-s}) = 0 \quad \not\rightarrow \quad \varepsilon_t \perp \varepsilon_{t-s}$$

ACF of squared epsilons : geometric decay

$$\varrho_{\varepsilon^2}(h) = \alpha_1^{|h|}$$



- Unconditional mean is zero
- Conditional mean is zero
- ACF of series shows no correlation with lags (just like stock returns)
- ACF of squared series does show correlation (just like stock returns)
- Allows us to model volatility clustering

GARCH(p,q) : persistent volatility, less HFT moves

ARCH(p) : AR(p)
GARCH(p,q) : ARMA(p,q)

There are other ways to form this analogy, but I think you get the idea...

$$\sigma_t^2 = \omega + \sum_{k=1}^{p} \alpha_k \varepsilon^2_{t-k} + \sum_{k=1}^{q} \beta_k \sigma^2_{t-k}$$

Persistent volatility

$$\varepsilon_t = z_t\sigma_t$$
$$\sigma_t^2 = \omega + \alpha_1\varepsilon^2_{t-1} + \beta_1\sigma^2_{t-1}$$

- Consider a GARCH(1,1) for simplicity (actually, this is the most common)
- ε_{t-1} term is random, σ_{t-1} term is not random
- The "randomness" comes from z_t
- Imagine $\beta_1 = 0.9$ - then σ_t would decay slowly over time

GARCH(1,1) as ARMA(1,1)

$$\eta_t = \varepsilon_t^2 - \sigma_t^2$$

$$\sigma_t^2 = \omega + \alpha_1\varepsilon^2_{t-1} + \beta_1\sigma^2_{t-1}$$
$$\sigma_t^2 = \omega + \alpha_1\varepsilon^2_{t-1} + \beta_1(\varepsilon^2_{t-1} - \eta_{t-1})$$
$$\sigma_t^2 = \omega + (\alpha_1 + \beta_1)\varepsilon^2_{t-1} - \beta_1\eta_{t-1}$$
$$\sigma_t^2 + \eta_t = \omega + (\alpha_1 + \beta_1)\varepsilon^2_{t-1} - \beta_1\eta_{t-1} + \eta_t$$
$$\varepsilon_t^2 = \omega + (\alpha_1 + \beta_1)\varepsilon^2_{t-1} - \beta_1\eta_{t-1} + \eta_t$$

$$\eta_t = \varepsilon_t^2 - \sigma_t^2$$
$$\sigma_t^2 = \omega + \alpha_1\varepsilon^2_{t-1}$$
$$\sigma_t^2 + \eta_t = \omega + \alpha_1\varepsilon^2_{t-1} + \eta_t$$
$$\varepsilon_t^2 = \omega + \alpha_1\varepsilon^2_{t-1} + \eta_t$$
$$y_t^2 = \omega + \alpha_1 y^2_{t-1} + \eta_t,\ \textit{where}\ y_t = \varepsilon_t$$

A deep learning approach to GARCH

1) How does it make predictions? (i.e. go from input to output)
   Usually some equation involving model parameters (ω, α, β for GARCH)
2) How do we find those model parameters?
   E.g. Deep Learning: “call fit” / “gradient descent”

- For deep learning, ARIMA, and many other ML models, the objective is based on the log-likelihood
- Maximum Likelihood Estimation (MLE): maximize the likelihood wrt model parameters - this is the “training process”
- Regression: typically minimize $\sum_i (y_i - \hat{y}_i)^2$ (squared error)
- Equivalent to maximizing likelihood when errors are normal
- This requires another assumption! The variance of errors is <u>constant</u>

Maximum likelihood in case of homoscedasticity

$$\mathit{Assumption}: \ y_t \sim \mathcal{N}(\hat{y}_t, \sigma^2)$$

$$\mathit{Equivalent}: \ y_t = \hat{y}_t + \varepsilon_t, \ \mathit{where} \ \varepsilon_t \sim \aleph(0, \sigma^2)$$

$$\hat{y}_t(\theta) = f(y_{t-1}, \ldots, y_{t-p}; \theta)$$

$$L(\theta) = \prod_{t=1}^{T} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\frac{(y_t - \hat{y}_t)^2}{\sigma^2}\right)$$

$$l(\theta) = \log L(\theta) = \sum_{t=1}^{T} \left\{-\frac{1}{2}\log(2\pi\sigma^2) - \frac{1}{2}\frac{(y_t - \hat{y}_t)^2}{\sigma^2}\right\}$$

$$\theta^* = \arg\max_{\theta} \sum_{t=1}^{T} \left\{-\frac{1}{2}\log(2\pi\sigma^2) - \frac{1}{2}\frac{(y_t - \hat{y}_t)^2}{\sigma^2}\right\}$$

$$\theta^* = \arg\max_{\theta} \sum_{t=1}^{T} \left\{-\frac{1}{2}\frac{(y_t - \hat{y}_t)^2}{\sigma^2}\right\}$$

$$\theta^* = \arg\max_{\theta} \sum_{t=1}^{T} \left\{-(y_t - \hat{y}_t)^2\right\}$$

$$\theta^* = \arg\min_{\theta} \sum_{t=1}^{T} \left\{(y_t - \hat{y}_t)^2\right\}$$

Max likelihood in case of hetero scedasticity

$$\theta^* = \arg\max_{\theta} \sum_{t=1}^{T} \left\{ -\frac{1}{2}\log(2\pi\sigma_t^{\,2}) - \frac{1}{2}\frac{(y_t - \hat{y}_t)^2}{\sigma_t^{\,2}} \right\}$$

$$\theta^* = \arg\max_{\theta} \sum_{t=1}^{T} \left\{ -\frac{1}{2}\log(\sigma_t^{\,2}) - \frac{1}{2}\frac{(y_t - \hat{y}_t)^2}{\sigma_t^{\,2}} \right\}$$

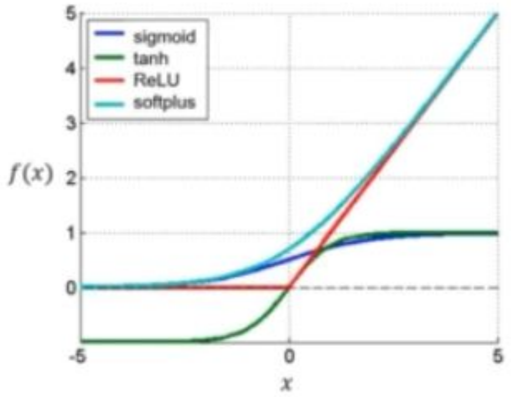
$$\theta^* = \arg\min_{\theta} \sum_{t=1}^{T} \left\{ \log(\sigma_t^{\,2}) + \frac{(y_t - \hat{y}_t)^2}{\sigma_t^{\,2}} \right\}$$

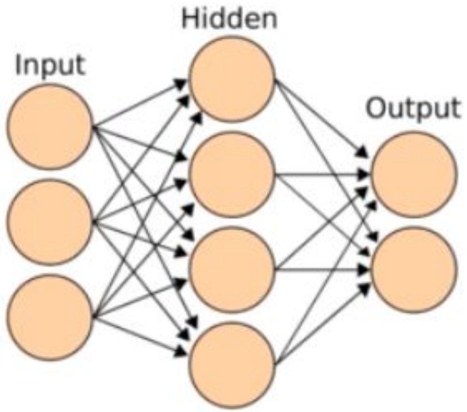
$$\theta^* = \arg\min_{\theta} \sum_{t=1}^{T} \left\{ \log(\sigma_t^{\,2}) + \frac{\varepsilon_t^{\,2}}{\sigma_t^{\,2}} \right\}$$

- Can plug into any stock optimizer (e.g. L-BFGS) - see Scipy docs
- We also need to include constraints on ω, α, β
- Easy to plug in with any optimizer
- Note: Derivation based on normal, but similar steps can be done for t-distribution, skewed t-distribution, etc.

The deep learning way

- Why stick with GARCH? (a linear model)
- We can parameterize σ_t using a neural network
- Just need to know how to make custom loss functions in TF2/PyTorch/...
- How to constraint output to be positive?
- Use positive-only activation (e.g. softplus)





Prerequisites for machine learning

Tensorflow 2.0

Tensorflow 2.0: Deep Learning and Artificial Intelligence

Lazy Programmer Inc., Lazy Programmer Team

PyTorch

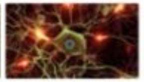
PyTorch: Deep Learning and Artificial Intelligence

Lazy Programmer Team, Lazy Programmer Inc.

In-depth deep learning series (up to CNNs and RNNs)

Data Science: Deep Learning and Neural Networks in Python

Lazy Programmer Inc.



Modern Deep Learning in Python

Lazy Programmer Inc.

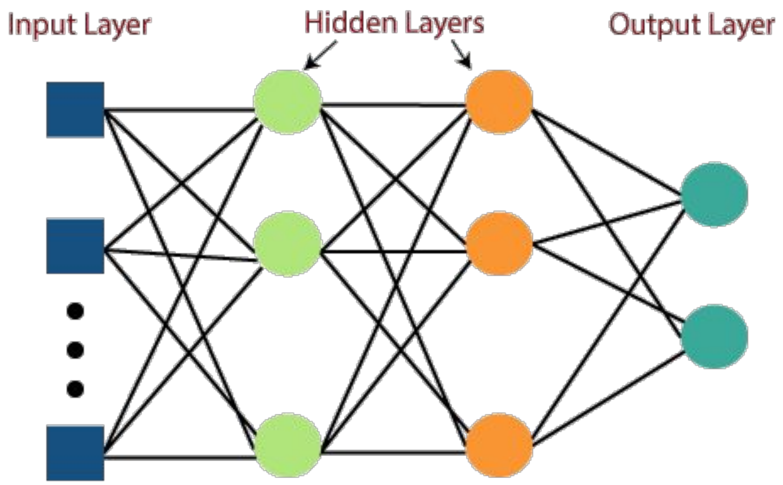
Deep Learning: Convolutional Neural Networks in Python

Lazy Programmer Inc.

Deep Learning: Recurrent Neural Networks in Python

Lazy Programmer Inc.

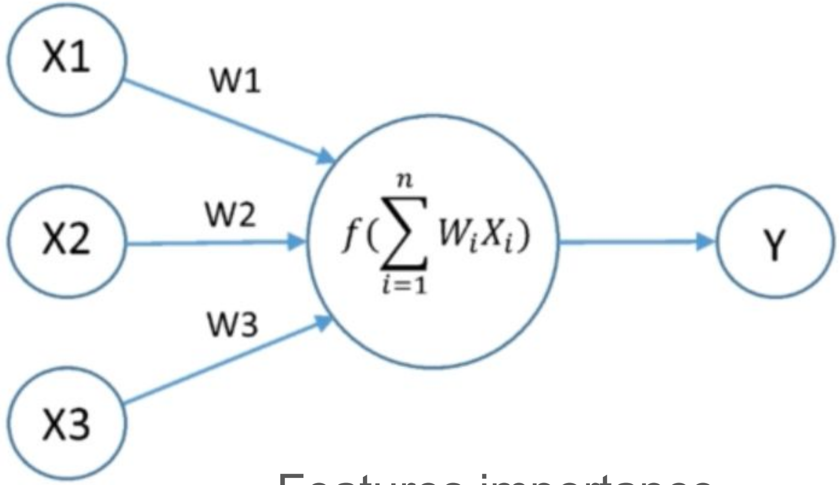
Multi-Layer perceptron



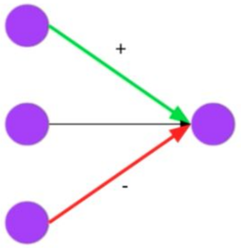
- https://github.com/fastai/course22/blob/master/04-how-does-a-neural-net-really-work.ipynb
- https://developers.google.com/machine-learning/crash-course/introduction-to-neural-networks/video-lecture?hl=fr

$$A\ line:\ ax + b$$

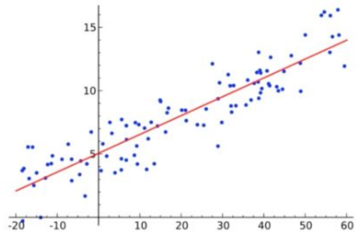
$$A\ neuron:\ \sigma(w^T x + b)$$



Features importance



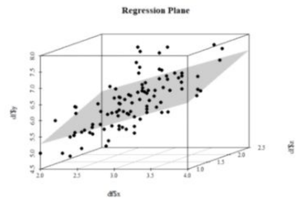
1d regression



$$\hat{y} = mx + b\ (or\ ax + b)$$

2d regression

$$\hat{y} = w_1 x_1 + w_2 x_2 + b$$

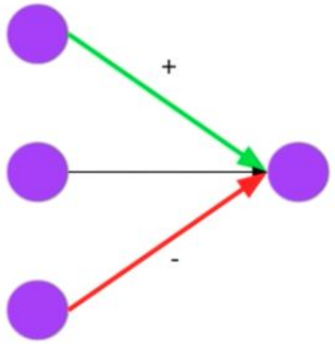


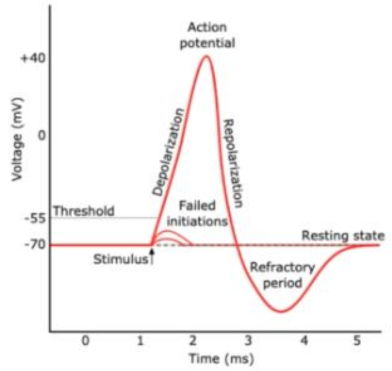
On the importance of introducing non-linearities through activation

- https://github.com/fastai/course22/blob/master/04-how-does-a-neural-net-really-work.ipynb
- Introduction aux réseaux de neurones (google)

The "threshold" in this case is -b (negative)

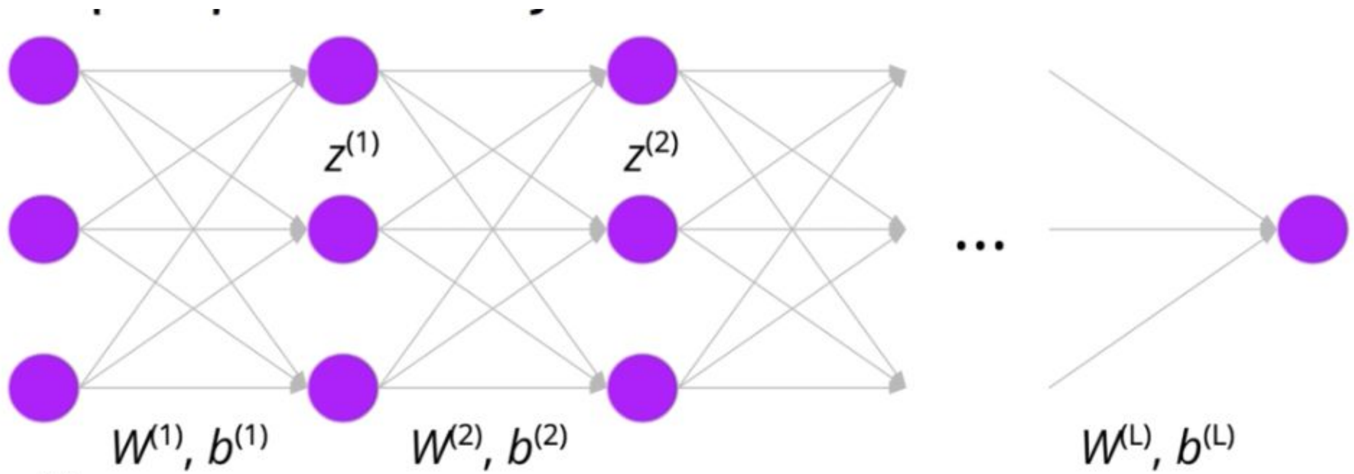
As long as $w^T x$ is > -b, then $w^T x + b > 0$, and the prediction is 1





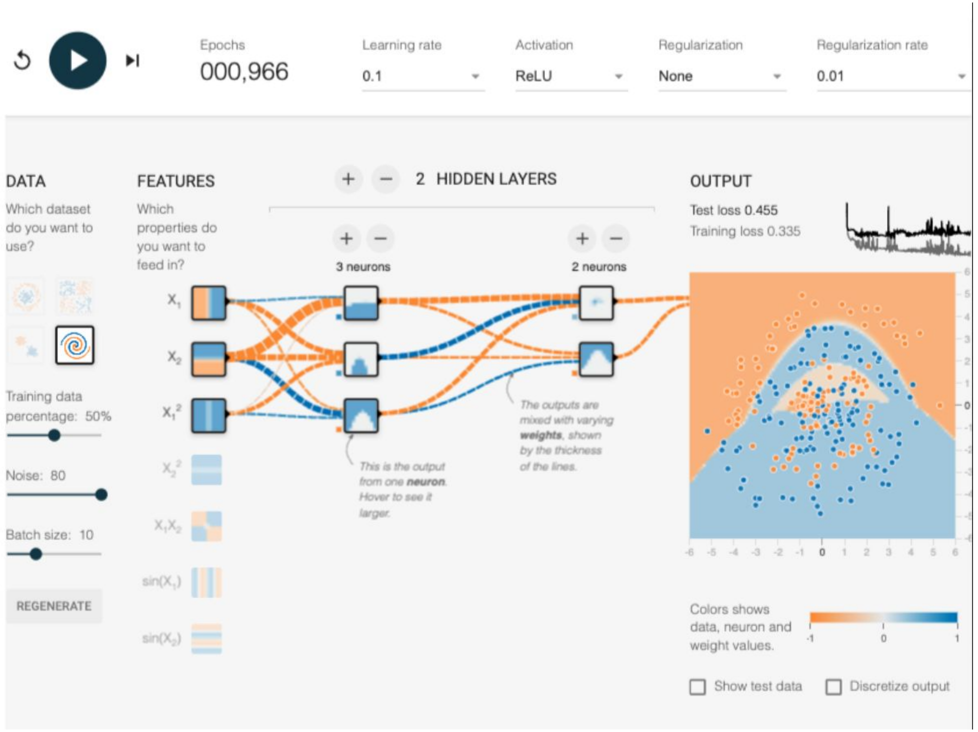
$$p(y = 1 \mid x) = \sigma(w^T x + b) = \sigma\left(\sum_{d=1}^{D} w_d x_d + b\right)$$

Stacking neurons



$$z^{(1)} = \sigma(W^{(1)T} x + b^{(1)})$$
$$z^{(2)} = \sigma(W^{(2)T} z^{(1)} + b^{(2)})$$
$$z^{(3)} = \sigma(W^{(3)T} z^{(2)} + b^{(3)})$$
$$\ldots$$
$$p(y = 1 \mid x) = \sigma(W^{(L)T} z^{(L-1)} + b^{(L)})$$

Fitting complex patterns through non-linear activation function and multiple layers



Epochs 000,966

Learning rate 0.1

Activation ReLU

Regularization None

Regularization rate 0.01

DATA

Which dataset do you want to use?

Training data percentage: 50%

Noise: 80

Batch size: 10

FEATURES

Which properties do you want to feed in?

2 HIDDEN LAYERS

OUTPUT

Test loss 0.455

Training loss 0.335

This is the output from one **neuron**. Hover to see it larger.

The outputs are mixed with varying **weights**, shown by the thickness of the lines.

Colors shows data, neuron and weight values.

Building MLP from scratch (tensorflow and pytorch)

- https://github.com/fastai/course22/blob/master/05-linear-model-and-neural-net-from-scratch.ipynb
- https://colab.research.google.com/github/google/eng-edu/blob/main/ml/cc/exercises/linear_regression_with_synthetic_data.ipynb

The matrix calculus you need for deep learning

- https://explained.ai/matrix-calculus