Data Analytics Final Prep

Chapter 1

Course Objectives
- Data Analytics terminology
- Design & apply models & systems
- Identify & explain Big Data Challenges

① Forecasting
- Time-series
- Linear Regression
- Support Vector Regression
- Neural Networks

② Deep Learning
- Network Architectures
- Convolution
- Pooling
- Recurrent networks
- Generative Adversarial networks

③ Recommender Systems
- Collaborative filtering
- Content-based filtering
- Context-aware
- Hybrid systems

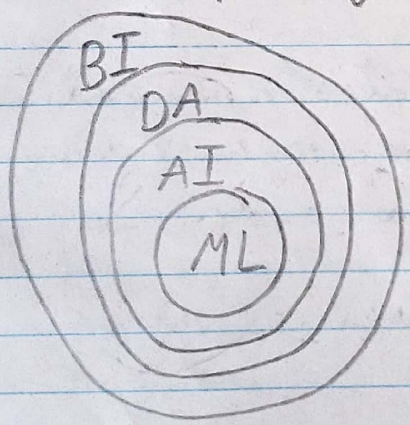
④ Big Data
- Challenges
- Approaches

Data Analysis
- Process of inspecting, cleansing, transforming & modeling data with the goal of discovering useful information, suggesting conclusions, and supporting decision making

- Business Intelligence
  - Applications, infrastructure & tools that enable access to and analysis of information to improve & optimize decisions & performance
  - Presents the insights determined by DA in a more business oriented way
  - DA sometimes considered part of BI

- Machine Learning
  - Science of getting computers to act without being explicitly programmed
  - Algorithms depend on some data to learn from

- Artificial Intelligence
  - Concept of machines being able to carry out tasks in a way we consider smart
  - ML is part of AI
  - ML & AI are parts of DA



- Data Science
  - Umbrella term for a group of fields that are used to mine large datasets.
  - Data Analytics is a more focused version of this: Realizes actionable insights that can be applied immediately based on existing queries

- Big Data
  - High volume, high velocity, and/or high variety information

assets that demand cost-effective innovative forms of information processing that enable enhanced insight, decision making, and process automation.
- 3Vs, 4Vs, ..., 10Vs?
- Volume: Amount, size & scale of data
- Variety: Structural variation, the data types it contains etc
- Velocity: Speed at which data are generated & rate to analyze it
- Veracity: Reliability of data

- Information extraction
  - Techniques to extract structured data from unstructured text

- Sentiment Analysis
  - Techniques to analyze opinionated text

- Social media Analytics
  - Content-based: Focuses on data posted by users
  - Structure-based: Focuses on analyzing relationships among participants

Chapter 2 - Forecasting

Focus: Choosing + Fitting model
Evaluating model

- Normally assumed Forecasting: Way in which the environment is changing will continue into the future

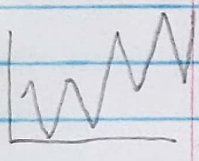
- Types
  - Qualitative: When past data is not available
  - Quantitative: Based on past data

- Time-series forecasting
  - Set of observations collected at different times at discrete & equally spaced time intervals

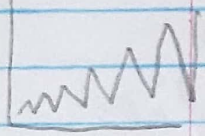
- Cross-sectional forecasting
  - Data on one or more variables collected at same point in time

- Time series patterns
  - Trend: Long term increase or decrease in data. Doesn't have to be linear
  - Seasonality: Series influenced by seasonal factors. Always of fixed & known period
  - Cyclic: Rises & falls that are not of fixed patterns. Duration usually 2 years

- Time Series Decomposition



  - Additive model: $\boxed{Y_t = S_t + T_t + E_t}$ [Seasonality + Trend (cyclic) + Error]
    e.g Increasing trend - still see same size peaks & troughs
    e.g Seen in indexed time series



  - Multiplicative model: $\boxed{Y_t = S_t \times T_t \times E_t}$
    e.g Increasing trend - amplitude increases - everything exaggerates
    e.g Web traffic
  - Seasonally adjusted data: Seasonal component removed from original data
    - Additive model: $\boxed{Y_t - S_t}$
    - Multiplicative model: $\boxed{Y_t / S_t}$

  - Moving averages: Classical method of time series decomposition
    - Estimate of trend-cycle at time t is obtained by averaging values of time series within K periods of t
      - Eliminates randomness, more smooth trend-cycle component.

Order m	$T_t = \frac{1}{m} \sum_{j=-K}^{K} y_{t+j}$	$m = 2K+1$

Order 5, K=2

Year	Sales	5-MA
1		
2		
3		x
4		y
5		z
6		
7		

x = average of sales of years 1-5
y = 2-6 ...

- Weighted moving averages: Combining moving averages
  - Smoother trend-cycle estimate
  - Weights of observations slowly increase & decrease instead of entering & leaving calculations at full weight

$$\boxed{T_t = \sum_{j=-K}^{K} a_j y_{t+j}} \quad \boxed{m = 2K+1}$$

$$\boxed{T_t = \frac{1}{8} y_{t-2} + \underbrace{\frac{1}{4} y_{t-1} + \frac{1}{4} y_t + \frac{1}{4} y_{t+1}}_{\text{Highest Impact}} + \frac{1}{8} y_{t+2}}$$

- Classical decomposition

Additive	Multiplicative
1 Calculate m-MA	Calculate m-MA
2 Calculate detrended series $y_t - T_t$	y_t / T_t
3 Calculate seasonal index thru avg $-S_t$	$\div S_t$
4 Remainder component	$E = y_t / S_t T_t$
$E_t = y_t - T_t - S_t$	

→ Group seasonal values → Avg^1 → Avg^2 of those averages

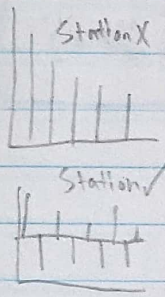
Index = Avg^1 / Avg^2

Value (− or ÷) S_t (index) = Seasonality removed

- ARIMA: Autoregressive integrated moving average
  - Another approach to time series forecasting
  - Aim to describe autocorrelations in data
  - No loss of observations at start and end of series
  - Seasonal component allowed to vary slowly over time

- A stationary time series is one whose properties do not depend on the time at which the series is observed
  - → No trend, no seasonality, constant variance
  - → Can be cyclic
  - → Mean & variance of graphs at different times should be similar

- Differencing: Differences between consecutive observations
  - → A way to make a time series stationary
  - → Differentiate until you get a stationary series
  - → Stables the mean of a time series by removing level changes eliminating or reducing trend & seasonality
  - → ACF for stationary drop to zero relatively quickly, unlike non-stationary that decrease slowly



- Autocorrelation: Correlation of a signal with a delayed copy of itself as a function of delay
  - → r_1 measures correlation y_t and y_{t-1}

$$\boxed{r_K = \sum_{t=K+1}^{T} \frac{(y_t - \bar{y})(y_{t-K} - \bar{y})}{\sum_{t=1}^{T} (y_t - \bar{y})^2}}$$

T: number of measurements

e.g. T=2 (min 2 measurements) $r_1 = \frac{\sum_{t=2}^{T} (y_2 - \bar{y})(y_1 - \bar{y})}{\sum_{t=1}^{T} (y_1 - \bar{y})^2}$

- Partial autocorrelation: Measure relationship between y_t and y_{t-K} after removing effects of other time lags 1, 2, 3, ... K-1
  - → Seeks to remove indirect correlations: Correlation of observation and observation at intervening time steps

$$\boxed{y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \phi_3 y_{t-3} + \dots + \phi_K y_{t-K} + e_t}$$

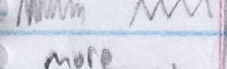
- Autoregressive models (AR)
  - → Forecast variable of interest using a linear combination of past values of the variable

AR(p) Model of order p

$$\boxed{y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \dots + \phi_p y_{t-p} + e_t}$$

- → Changing ϕ_p results in different time series patterns
- → Autoregression : Regression of variable against itself
- → Changing variance of e_t changes scale of series & not patterns

AR(1) AR(2)



more spaced out

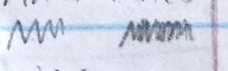
- Moving average models (MA)
  - → Uses past forecast errors rather than values in a regression-like model

MA(q) Model of order q

$$\boxed{y_t = c + e_t + \theta_1 e_{t-1} + \theta_2 e_{t-2} + \dots + \theta_q e_{t-q}}$$

- → Changing θ_q result in different time series patterns
- → We do not observe values e_t
- ✱ Moving average model is used for forecasting future values while moving average smoothing is used for estimating trend cycle of past values

MA(1) MA(2)



less spaced out

- Autoregressive Moving averages (ARMA) models
  - → Autoregression + Moving average
  - → Stationary series. Has no integrated components

ARMA (p,q)

$$\boxed{X_t = c + \varepsilon_t + \sum_{i=1}^{p} \phi_i X_{t-i} + \sum_{i=1}^{q} \theta_i e_{t-i}}$$

- → Used to describe weakly stationary stochastic (random) time series
- → p is the order of autoregressive polynomial
- → q is the order of moving average polynomial

- Backshift notation : Effects of shifting data one period
  → Describes the process of differencing

$$\boxed{B y_t = y_{t-1}}$$

Shifts data back one period
$B^2 y_t$ = 2 periods

First difference $(1-B) y_t$
Second difference $(1-B)^2 y_t$

→ Particularly useful when combining differences

- Autoregressive Integrated Moving Average (ARIMA) models
  → Differencing + Autoregression + Moving average
  → I stands for integrated; measure of how many non-seasonal differences are needed to achieve stationarity

ARIMA (pdq) model:

$$\boxed{y_t' = c + \phi_1 y_{t-1}' + \dots + \phi_p y_{t-p}' + \theta_1 e_{t-1} + \dots + \theta_q e_{t-q} + e_t}$$

→ y_t' is the differentiated series

$$\boxed{\underbrace{(1 - \phi_1 B - \dots - \phi_p B^p)}_{AR(p)} \underbrace{(1-B)^d}_{d \text{ differences}} y_t = c + \underbrace{(1 + \theta_1 B + \dots + \theta_q B^q)}_{MA(q)} e_t}$$

→ d is the degree of first differencing involved

→ How to select p, d, q? Possible approach: Maximum Likelihood

ACF
PACF

→ ARIMA $(p, d, 0)$ if
- ACF exponentially decaying
- Significant spike at lag p in PACF, but none beyond

ARIMA $(0, d, q)$ if
- PACF exponentially decaying
- Significant spike at lag q in ACF, but none beyond

→ ARIMA Forecasting - point forecasts
- ARIMA(3,1,1) example

$B y_t = y_{t-1}$

① $$(1-\phi_1 B-\phi_2 B^2-\phi_3 B^3)(1-B)y_t = (1+\theta_1 B)e_t$$
$$(1-\phi_1 B-\phi_2 B^2-\phi_3 B^3)(y_t - y_{t-1}) = e_t + \theta_1 e_{t-1}$$
$$y_t - \phi_1 y_{t-1} - \phi_2 y_{t-2} - \phi_3 y_{t-3} - y_{t-1} + \phi_1 y_{t-2} + \phi_2 y_{t-3} + \phi_3 y_{t-4} = e_t + \theta_1 e_{t-1}$$
$$y_t = y_{t-1}(1+\phi_1) - y_{t-2}(\phi_1-\phi_2) - y_{t-3}(\phi_2-\phi_3) - y_{t-4}\phi_3 + e_t + \theta_1 e_{t-1}$$

$H=1$
T = last observation

② $$y_{T+1} = (1+\phi_1)y_T - (\phi_1-\phi_2)y_{T-1} - (\phi_2-\phi_3)y_{T-2} - \phi_3 y_{T-3} + e_{T+1} + \theta_1 e_T$$

Replace future observations & past errors & future errors = 0

③ $$y_{T+1|T} = (1+\phi_1)y_T - (\phi_1-\phi_2)y_{T-1} - (\phi_2-\phi_3)y_{T-2} - \phi_3 y_{T-3} + \theta_1 \hat{e}_T$$

e.g. $$y_{T+2|T} = (1+\phi_1)y_{T+1|T} - (\phi_1-\phi_2)y_T - (\phi_2-\phi_3)y_{T-1} - \phi_3 y_{T-2}$$

- Log transform data is one of the best ways to make a series stationary on variance
- If differencing of model doesnot produce stationary results on variance, then log transform data and difference that. Produces stationary mean and variance

Evaluation & Accuracy measures

- Forecast error: $\boxed{e_i = y_i - \hat{y}_i}$

  observation, forecast

Scale Dependant Errors

- Mean Absolute Error (MAE): Mean of absolute values of error

$$\boxed{MAE = \frac{1}{n}\sum_{j=1}^{n} |y_j - \hat{y}_j|}$$

- Mean Squared Error (MSE): Focuses on large errors

$$\boxed{MSE = \frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}$$

- Root Mean Squared Error (RMSE): Interpretable in the same unit as response vector or y units making it easy to relate its info

$$\boxed{RMSE = \sqrt{MSE}}$$

Percentage error

- Mean absolute percentage error (MAPE): Can't divide by 0!

$$\boxed{MAPE = \text{mean}(|p_i|) = \frac{100\%}{n}\sum_{j=1}^{n}\left|\frac{y_j - \hat{y}_j}{y_j}\right|}$$

- Evaluation is how well a model performs on new data not used when fitting the model and not the historical data

- Train/Test → Usually 80/20 → Insample data / Hold-out set
  Out of sample data

- Cross-validation

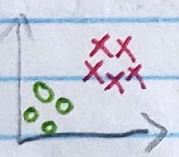
e.g 5-fold cross validation
- Split training set into 5 subsets
- Retain one subset for validation and train with remaining subsets
- Evaluate error on validation / test subset
- Repeat process 5 times with different validation subsets
- Calculate overall error for 5 folds

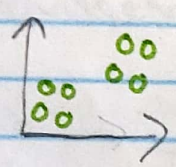
Train	Train	Train	Test	Train

Chapter 3 - Forecasting LR + SVR

- Supervised learning: Predicting based on labeled data
  → Forecasting

  Unsupervised learning: Predicting based on unlabeled data
  → Need to find structure within data
  → Clustering


Supervised


Unsupervised

- Supervised
  - Classification: Discrete class labels or categories output
  - Regression : Continuous values output

- Regression: Approach for modelling relationship between dependent variable and independent variables

- Linear Regression: Linear relationship between a dependent variable and independent variables
  → Least squares estimation

$$\boxed{y = \beta_0 + \beta_1 x + \varepsilon}$$

$y = \theta_1 x + \theta_0$

- Choose θ_0 & θ_1 to minimize sum of squared errors

$$\boxed{\sum_{i=1}^{N} e_i^2 = \sum_{i=1}^{N} (y_i - \theta_0 - \theta_1 x_i)^2}$$

or in Andrew Ng's notes $\boxed{\frac{1}{2m} \sum_{i=1}^{m} (\overset{\theta_1 x + \theta_0}{h_\theta}(x^{(i)}) - y_i)^2}$

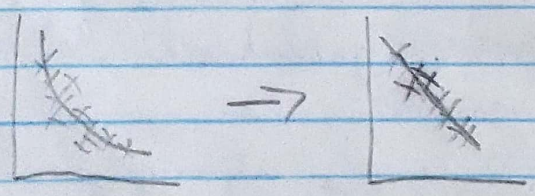
Cost / Objective function!

$$\boxed{\theta_1 = \frac{\sum_{i=1}^{N} (y_i - \bar{y})(x_i - \bar{x})}{\sum_{i=1}^{N} (x_i - \bar{x})^2}} \qquad \boxed{\theta_0 = \bar{y} - \theta_1 x}$$

- Linear regression assumptions and conditions
  - → No (or little) multicollinearity - independent variables not highly correlated with each other
  - → Multivariate normality - residuals are normally distributed
  - → Homoscedasticity - Residuals are similar across the independent variables (across regression line)

- Non-linear functional forms
  - → Transform variables x and/or y then estimate regression model
  - → Most common - logarithmic

Log-log form $\boxed{\log y_i = \theta_1 \log x + \theta_0 + \varepsilon_i}$



Multivariate linear regression

K = number of features
x_{ij} = value of feature j of the ith training example

$$\boxed{y_i = \theta_0 + \theta_1 x_{1,i} + \theta_2 x_{2,i} + \dots + \theta_K x_{K,i} + e_i}$$

$$\boxed{y_i \text{ (or } h_\theta(x)) = \theta^T x + e}$$

Cost function $\boxed{(Y - X\theta)^T (Y - X\theta)}$

$$\boxed{\theta = (X^T X)^{-1} X^T Y}$$ Normal equation

Goodness of line fit

- Coefficient of determinant R^2 : How close data are to the fitted line
  How well model fits the data

Relative

$$\boxed{R^2 = \frac{\sum (\hat{y}_i - \bar{y})^2}{\sum (y_i - \bar{y})^2}}$$

predicted, observed, mean

$\frac{\text{Explained variation}}{\text{Total variation}}$ 0-1 range

0% : Model explains none of the variability of the response data around its mean
100% : Model explains all of the ''' '''
Fitted values would always = the observed values

e.g. $R^2 = 0.50$ → Approx half of the observed variation can be explained by model

- Standard error of regression : Absolute measure of typical distance that the data points fall from the regression line
  → Standard deviation of the residuals

Absolute

s is in the units of the dependent variable

$$\boxed{S_e = \sqrt{\frac{1}{N-2} \sum_{i=1}^{N} e_i^2}}$$

N-2 cause sample & slope + intercept were estimated to estimate sum of squares

Prediction Intervals

- Describe the uncertainty for a single specific outcome
- An interval which we expect y_i to lie within, with a specified probability

$$\boxed{\hat{y}_t \pm K\hat{\sigma}}$$

- K: multiplier - corresponds to percentage. 95% : $K = 1.96$
- $\hat{\sigma}$: forecasted standard deviation estimate

- Assuming regression errors are normally distributed, 95% forecast interval

$$\boxed{\hat{y} \pm 1.96\, s_e \sqrt{1 + \frac{1}{N} + \frac{(x-\bar{x})^2}{(N-1)sd^2}}}$$

sd of observed values x

Classifier

Support Vector Machine

→ The best choice will be the hyperplane that leaves the maximum margins from both classes

→ Will have the highest robustness to noise

$$\boxed{y_n = W^T X_n + b}$$

Weights & Normal to hyperplane — Input — Bias

$y = \{-1, 1\}$ ← Class Labels

$y_n(W^T X_n + b) > 0$

$y(z) = 1$ if $z \geq 0$

$y(z) = -1$ otherwise

$$\boxed{W^T X_n = -b \text{ on hyperplane}}$$

Unit Vector $\frac{W}{\|W\|}$

$$\boxed{\text{min distance to closest points from margin} = \frac{1}{\|W\|} \min y_n (W^T X_n + b)}$$

Margin // Condition //

Maximizing margin

① maximize $\frac{2}{\|W\|}$

subject to: $\min_{n=1,\dots,N} y_n(W^T X_n + b) \geq 1$

② minimize $\frac{1}{2} W W^T$

subject to: $y_n(W^T X_n + b) \geq 1$ for $n = 1, \dots, N$

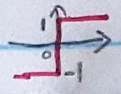
- Example $X = \begin{bmatrix} 0 & 0 \\ 2 & 2 \\ 2 & 0 \\ 3 & 0 \end{bmatrix}$ $y = \begin{bmatrix} -1 \\ -1 \\ 1 \\ 1 \end{bmatrix}$ subject to ②

1st row $-1(W_1 \cdot 0 + W_2 \cdot 0 + b) \geq 1$
2nd $-1(W_1 \cdot 2 + W_2 \cdot 2 + b) \geq 1$
3rd $1(W_1 \cdot 2 + W_2 \cdot 0 + b) \geq 1$
4th $1(W_1 \cdot 3 + W_2 \cdot 0 + b) \geq 1$

$-b \geq 1$, $-2W_1 - 2W_2 - b \geq 1$, $2W_1 + b \geq 1$, $3W_1 + b \geq 1$

$b \leq -1$ | $W_1 \geq 1$? | $-2W_2 \geq 2$, $W_2 \leq -1$ | $W_1 \geq 1$? | $3W_1 - 1 \geq 1$, $W_1 = 2/3$?

Signum
-1 if $x<0$
0 if $x=0$
1 if $x>0$



either $\boxed{\begin{matrix} W_1 = 1 \\ W_2 = -1 \\ b = -1 \end{matrix}}$ or $\boxed{\begin{matrix} W_1 = 2/3 \\ W_2 = -1 \\ b = -1 \end{matrix}}$

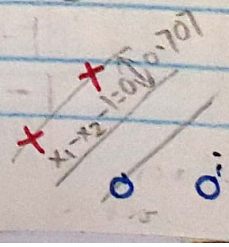
So $\frac{1}{2}(W_1^2 + W_2^2) \geq 1$

$(W^T X_n + b)$
$y_n(1 \cdot x_1 - 1 \cdot x_2 - 1) = y$
$y_n(x_1 - x_2 - 1) = y$

Optimal Hyperplane
- $g(x) = \text{sign}(x_1 - x_2 - 1)$

margin: $\frac{1}{\|W^*\|} = \frac{1}{\sqrt{2}} \approx 0.707$

For ①, ②, ③
$y_n(W^{*T} x_n + b^*) = 1$
↑
Support vectors



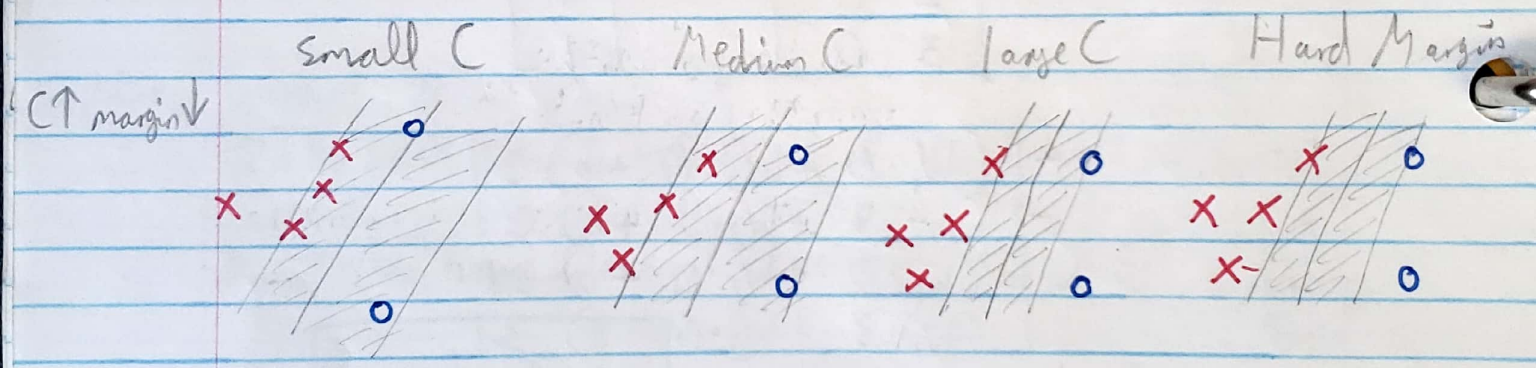
- Support vectors are datapoints on boundaries
- Quadratic programming to solve SVM

- Separable data - hard margin SVM
- Non-separable data - soft margin SVM
  - → Allow some points to lie on other side
  - → Tolerates error

$$\boxed{\begin{array}{l} \text{minimize } \frac{1}{2} W W^T + C \sum_{n=1}^{N} \xi_n \\ \text{subject to } y_n (W^T x_n + b) \geq 1 - \xi_n \qquad \xi_i \geq 0 \text{ for } n = 1, \ldots, N \end{array}}$$

→ If an example margin is within boundary ($\xi_i > 0$) we pay a cost of objective function being increased by $C \xi_i$

→ C (cost) balances between making the margin large and ensuring most examples are outside of the margin



- If data is linearly inseparable → Feature transform

e.g $X = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$ $\quad Z = \Phi(x) = \begin{bmatrix} x_1^2 \\ x_2^2 \end{bmatrix} = \begin{bmatrix} 1 \\ \phi_1(x) \\ \phi_2(x) \end{bmatrix} = \begin{bmatrix} 1 \\ z_1 \\ z_2 \end{bmatrix}$

1. Original data
2. Transform data into Z-space ← Φ
3. Separate data in Z-space
4. Classify in X-space ← Φ^{-1}

- Kernels: Transform non-linearly separable without physically transforming data to Z-space
  - → Can be written in terms of the inner product
  - → Kernel helps calculate inner product in Z-space
  - → Computationally cheaper than explicit computation of coordinates

eig 2nd-order polynomial transform

$$\Phi(x) = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix} \qquad K(x,x') = \Phi(x)^T \Phi(x') = \begin{bmatrix} \\ \\ \\ \\ \\ \end{bmatrix} \cdot \begin{bmatrix} \\ \\ \\ \\ \\ \end{bmatrix}$$

$O(d^2)$
↑ dimension number

$$= x_1x_1' + x_2x_2' + x_1^2x_1'^2 + 2x_1x_2x_1'x_2' + x_2^2x_2'^2$$

$$= \left(\tfrac{1}{2} + x^Tx'\right)^2 - \tfrac{1}{4}$$ → Computed quickly in X-space, in $O(d)$

→ Popular kernels: Polynomial, hyperbolic tangent, Gaussian RBF

Gaussian RBF → $$\boxed{K(x,x') = \exp\left(-\frac{\|x-x'\|^2}{2\sigma^2}\right) = \exp\left(-\gamma\|x-x'\|^2\right)}$$

- Support Vector Regression (SVR) - for regression

$$\boxed{\text{minimize } \frac{1}{2}\|W\|^2 \text{ subject to } \begin{cases} y_i - \langle w, x_i\rangle - b \le \varepsilon \\ \langle w, x_i\rangle + b - y_i \le \varepsilon \end{cases}}$$

x_i training sample with target y_i
$\langle w, x_i\rangle + b$ prediction for x_i

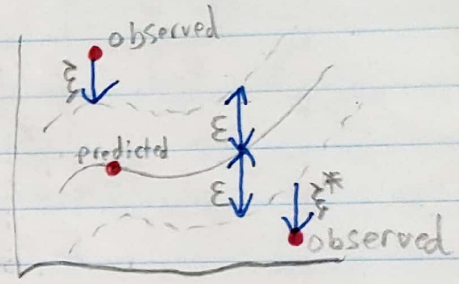
- ε is a free parameter that serves as a threshold: all predictions have to be within an ε range of the true predictions

- Slack variables are usually added to allow for errors

$$\text{Minimize } \frac{1}{2}\|w\|^2 + C\frac{1}{N}\sum_{i=1}^{N} \xi_i + \xi_i^*$$

$$\xi_i, \xi_i^* \geq 0$$

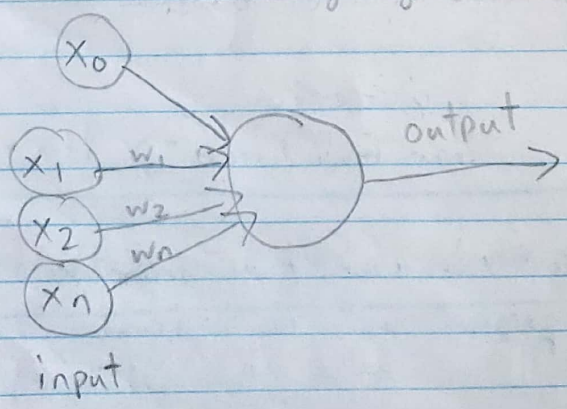
ξ_i & ξ_i^* capture residuals beyond ε boundary



- Data snooping - Avoid! Determine input transformation of test set from training data only.

Chapter 4 - Neural Networks

- Perceptron: Computer model that simulates the brain's ability to recognize and discriminate

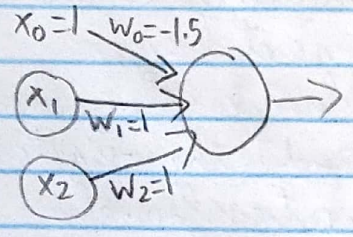


$$net = \sum_{i=1}^{n} w_i x_i$$

$$y = \begin{cases} 1 & \text{if } net > 0 \\ 0 & \text{otherwise} \end{cases}$$

→ A single layer feed-forward neural network with preprocessing

- Boolean AND

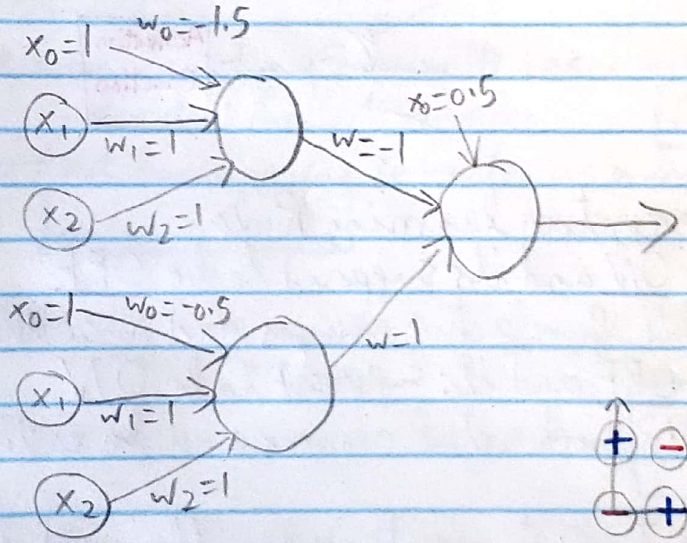


x1	x2	output
0	0	0
0	1	0
1	0	0
1	1	1

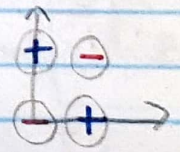
$1\cdot1+1\cdot1-1.1.5>0$

- Boolean OR $\quad w_0=-0.5$ instead of -1.5

- Boolean XOR (Exclusive OR)



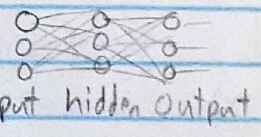
x_1	x_2	output
0	0	0
0	1	1
1	0	1
1	1	0



Not linearly separable!

→ Multi-layer perceptron to solve Boolean XOR.
- Increases expressiveness of the network

Multi-layer perception
e.g 3-layer MLP



→ Any target function f that can be decomposed into linear separators can be implemented by a 3-layer MLP

→ MLP size ↑ = Approximation ↑ Generalization ↓

$y_j = \sum_{i=1}^{K} w_{ij} x_i \qquad Y_j = W_j X_j \qquad Y_j = W_j X_j + b$

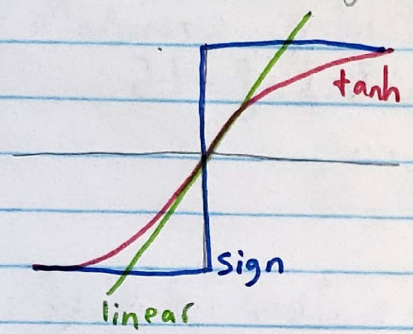
y = output
x = neuron inputs
w = weights
b = bias

Activation Function

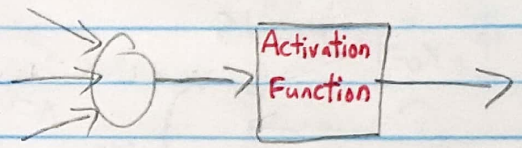
→ Non linear transformation done over the input signal. Transformed output then sent to next layer

→ Make backpropagation possible since the gradients are supplied alongwith the error to update the weights & biases

→ No activation function = Linear transformation



- sign → not smooth → cannot use gradients
- tanh is smooth; can be used



Multi-layer perceptron learning Rule

- The output is ON and its supposed to be OFF
  → Lower weights from active neurons and raise the threshold
- The output is OFF and its supposed to be ON
  → Raise weights from active neurons and lower the threshold

Gradient Descent: Optimization algorithm used to minimize some function by iteratively moving in the direction of steepest descent as defined by the negative of the gradient

$$\boxed{\frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)})-y^{(i)}\right)^2}$$

e.g When we move the ball a small amount Δv_1 in v_1, and Δv_2 in v_2

C Function changes	$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2$
vector of changes	$\Delta v \equiv (\Delta v_1, \Delta v_2)^T$
Gradient vector	$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}\right)^T$
	$\boxed{\Delta C \approx \nabla C \cdot \Delta v}$ → Gradient vector relates vector of changes to function C changes

- Batch vs Mini-Batch vs Stochastic gradient descent
  → Andrew Ng's notes

Backpropagation
→ Method of computing gradients in directed graphs of computation such as neural networks
→ It's about understanding how changing weights & biases in a network changes the cost function
→ Basically to minimize cost function

- Assumptions
  → Cost function can be written as an average over cost functions for individual training samples $\boxed{C = \frac{1}{n} \sum_x C_x}$
  → Cost function can be written as a function of network outputs
  $\boxed{\text{cost } C = C(a^L)}$

- Activation of neuron j in layer l $\boxed{a^l = \sigma(w^l a^{l-1} + b^l)}$

- Error of neuron j in layer l $\boxed{\delta_j^l \equiv \frac{\partial C}{\partial z_j^l}}$ ← weighted input

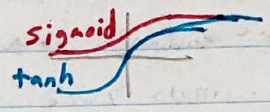
- Backpropagation technique & equations → Andrew Ng's notes

- Training a neural network

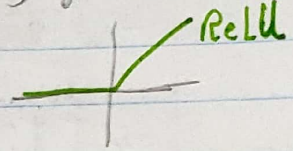
1. Input set of training examples
2. For each training example x
   - Feedforward
   - Output error
   - Backpropagate error
3. Gradient descent

- A change in some weight in the network will cause a change in the output activation from the corresponding neuron, causing changes in all activations in next layer, and next and next so on, changing final layer & cost funct

- Sigmoid $S(x) = \frac{1}{1+e^{-x}}$ Tanh deals easier with negative numbers

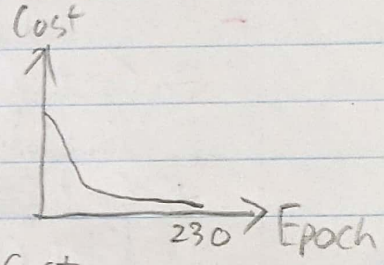


Rectified linear (units) (ReLU): Solves vanishing gradient problem

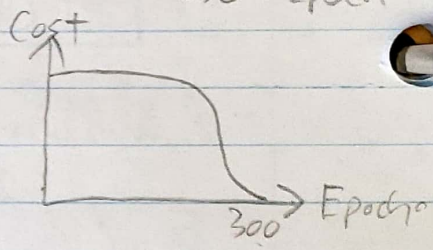


- How fast NN learns depends on initial weights
  → At the beginning, learning rate will be steep due to large errors

- e.g $w=0.6$, $b=0.9$ init output $=0.82$



- $w=2$, $b=2$, init output $=0.98$



→ Why does it get slow? Rate of change depends on derivatives, and $\sigma'(z)$ gets very small when output close to 1
sigmoid, tanh etc

- Cross-entropy cost function
  → Measures performance of a classification model whose output is a probability value between 0 and 1
  → Loss increases as predicted probability diverges from actual label
  → Perfect model = 0 log loss
  → Rate at which weights learn is controlled by the error in the output

$$\boxed{C = -\frac{1}{n}\sum_x \left[y \ln a + (1-y)\ln(1-a)\right]}$$

- When neuron is unambiguously wrong
  → Cross-entropy learns faster than quadratic cost
- Cross-entropy often better if output neurons are sigmoid

- SoftMax activation + log-likelihood
  → Another way to deal with slowdown problem

Soft Max
  → Defines a new type of output layer for our neural networks
  → Output - a set of positive numbers that sum up to 1
    Basically a probability distribution
  → weighted inputs ↑ Corresponding output activation ↑

- Example : Input image 7 Log-likelihood cost $-\ln a_7^L$
  → Network doing good job → a_7^L prob close to 1 → $-\ln a_7^L$ small

→ Log-likelihood acts like a cost function

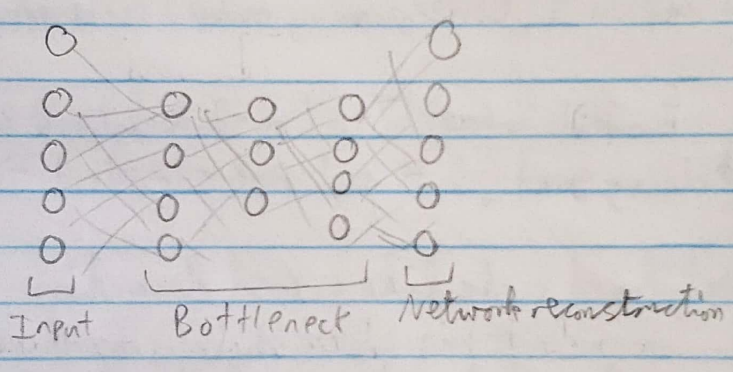
→ Softmax + log-likelihood similar to Sigmoid + Cross-entropy

Chapter 5 - Deep Learning

- Deep learning : Learning system with several non-linear transformations
- Difference between deep learning & feed-forward multilayer networks:
  1) More neurons than previous networks
  2) Explosion in the amount of computing power available to train
  3) More complex ways of connecting layers/neurons in NN
  4) Automatic feature extraction
     → Feature engineering : • Creating new features using domain knowledge
       • Handcrafted
     → Feature learning : • System automatically discovers features
       • Takes complex raw data & creates higher-order features automatically
       e.g Image sharpening, image captioning automatically

- → FFNN - layers fully connected to neighboring layers
- → CNN - locally connected patches of neurons between layers
- → RNN - recurrent connections to the same neuron + connections from prev layers

- Unsupervised networks - Autoencoders
  - → Used for representational learning [learn representations of dataset]
  - → Output is a reconstruction of the input
  - → Often part of large network
  - → Same number of outputs as inputs → Hidden layer fewer than input



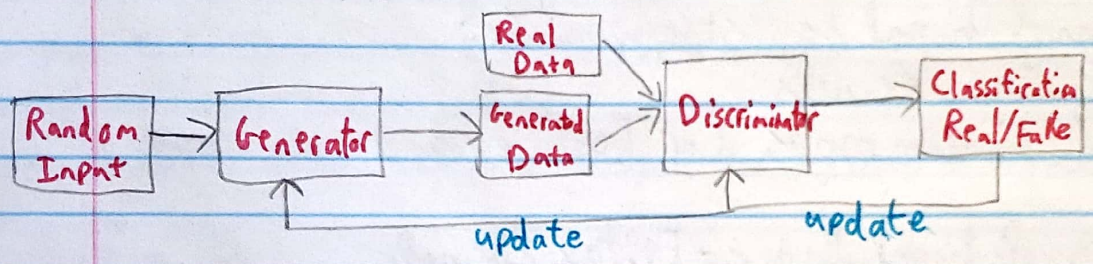
→ Unsupervised as data is unlabeled, but has supervised elements as there are expected (number of) outputs

- Uses:
  - → Dimensionality reduction, feature extraction/selection
  - → Anomaly detection
  - → Denoising autoencoders
  - → Variational autoencoders e.g. For generative tasks (like GAN)

- Unsupervised networks - Generative Adversial Networks
  - → Synthesizing novel new signals/images based on other training images e.g. Generating new images, sound, video
    Generating images from text descriptions

GAN : Generative network + Discriminator network

- Generative network
  - → Generates data/images by discovering patterns in training data
  - → Training GANs - update parameters such data that network will generate more believable images & fool discriminator
- Discriminator network
  - → Attempts to classify output as real or synthetic
  - → Trained on samples from dataset
  - → Typically a CNN



- GANs are hard to train [Training divergence, mode collapse, vanishing gradients]

Convolutional Neural Network (CNN)

→ Learn higher-order features in the data via convolutions

→ Mainly used with object recognition with images (Computer vision)

- Input data
  - Width : Image width in pixels
  - Height : Image height in pixels
  - Depth : RGB channels

e.g. 32x32 pixels 3 RGB channels = 3072 input neurons

Progressively construct higher-order features!

- Convolutional layers
  - → Transform input data by using a patch of locally connecting neurons from the previous layer
  - → Neurons in layer are connected to only a small region of neurons in the layer before it
  - → Compute dot product between region of input neurons and weights to which they are connected
  - → Progressively construct higher-order features

- Convolutional layers components:
  1) Filters / Kernels
  2) Activation maps (or Feature map)

- Convolutional layers - Filters / Kernels
  - → Sets of weights in a convolutional layer
  - → Result = Feature map [Filter x Input]
  - → Applied over width, depth, and height
  - → Filter count is a hyperparameter. It controls how many activation maps are produced
  - → Learned filters produce strongest activation to spatially local input patterns

- Convolutional layers - Activation map
  - → Sliding each filter across width and height produces activation map
  - → Slide filter across input volume depth
  - → stacking activation maps along depth dimension in output
  - → Each neuron generating the output volume is connected to only a local region of the input volume

- Convolutional layers - Activation map - Receptive field
  - → Controls local connectivity
  - → How much of the input width and height connects to neurons
  - → Connectivity along depth axis is always full

e.g 32x32x3 image 5x5 receptive field

- Weights for each neuron in convolution layer? $5 \times 5 \times 3 = 75$
- Weights with full connectivity? $32 \times 32 \times 3 = 3072$

- ReLU commonly used with CNN
  → Changes output, not dimensions

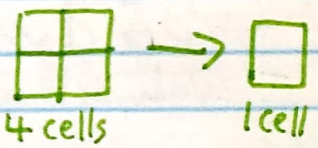
- Pooling layers
  → Downsampling operation
  → Progressively reduces spatial size
  → Controls overfitting
  → Operates independently on every depth slice

e.g 2x2 filter, max() takes largest of four numbers

e.g 32x32 image, 2x2 filter with stride 2

- Size reduction in percentage?

$$C = \frac{I - F + 2P}{S} + 1$$



75% reduction

S = stride
P = Padding
C = convoluted matrix
I = input matrix
F = filter matrix

CNN examples

- AlexNet & ResNet used for classifying images
  → AlexNet computationally expensive & time consuming to train
  → ResNet had skip connections or gated recurrent units
    → Reduced computation complexity
    → Outperformed AlexNet

- Recurrent Neural Networks (RNN)
  → In the family of FFNN
  → Models time dimension by creating cycles

- SVR, NN, Logistic regression does not model in time dimension
  → Can capture time using sliding window

- RNN include a feedback loop to learn from sequences
  → Good at modelling temporal behaviors - time series, language audio, text

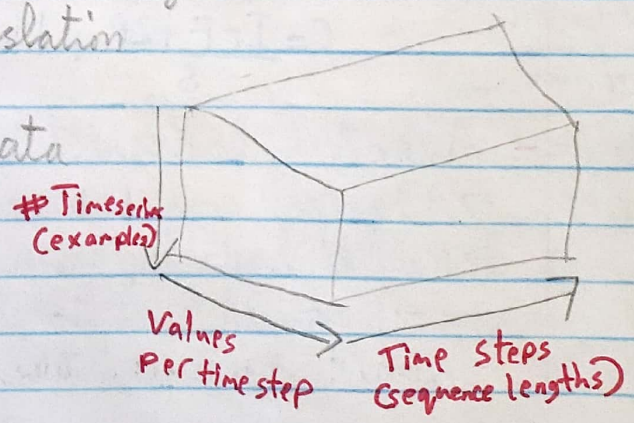
- RNN trained to generate sequences. Output at each time step based on:
  - Current input
  - Input at all previous time steps
  → Computes a gradient with an algorithm called backpropagation through time (BPTT) (computes cost)

e.g Inputs & outputs

1 - many	many - 1
- Speech synthesis	- Sentiment analysis
- Music generation	- Time-series prediction

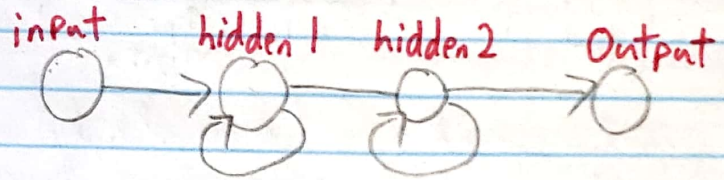
many - many
- Video classification: label each frame
- Natural language translation

- Recurrent Network Data



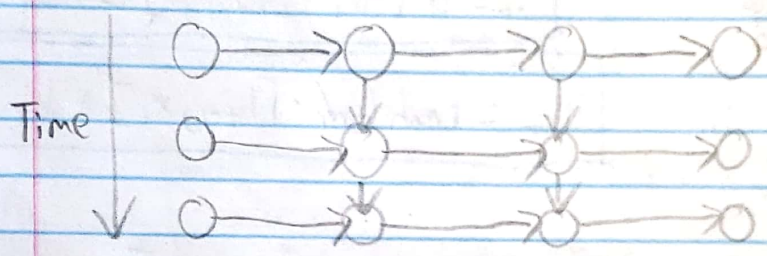
Long Short-Term Memory (LSTM) networks
→ Main components:
- Memory cells
- Gates: input, forget and output gates
→ Gating structure allows
- Information to be retained across many time steps
- Gradients to flow across many time steps (dealing with vanishing gradient problem)

→ Explicitly designed to avoid long-term dependency problem



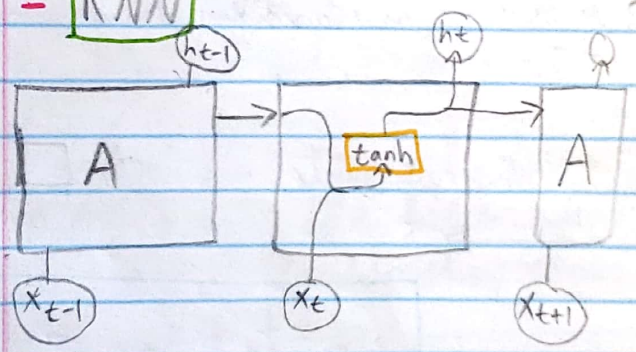
·4 layers example

If we unfold it



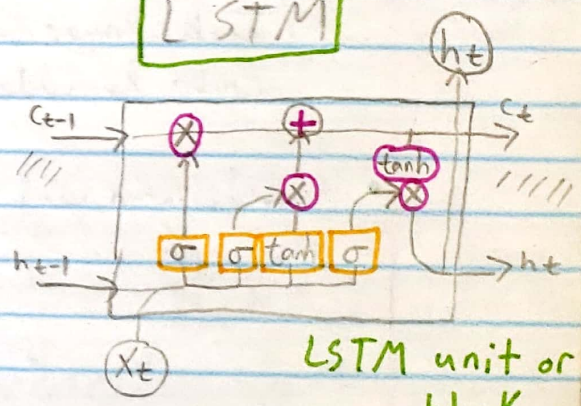
- RNN

3 units in 1 layer



vs

LSTM

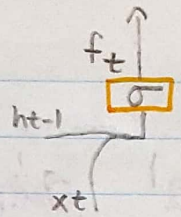


LSTM unit or block

□ Neural Network Layer

→ Gates are a way to optionally let information through
→ LSTM units has 2 connections
  1) From previous time-step
  2) From previous layer
→ Three gates
  1) Input gate - Protects the unit from irrelevant events
  2) Forget gate - Decides what information to throw from previous memory contents
  3) Output gate - Exposes the content of memory cell at LSTM unit output

- Forget gate

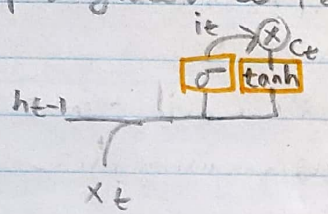


$$\boxed{f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)}$$

weight bias

→ Sigmoid function, so it output a number between 0 and 1

- Input gate & cell state

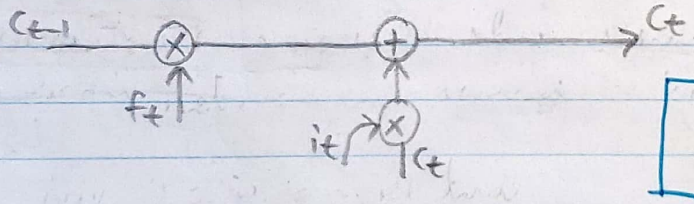


$$\boxed{i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)}$$

$$\boxed{\tilde{C}_t = \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right)}$$

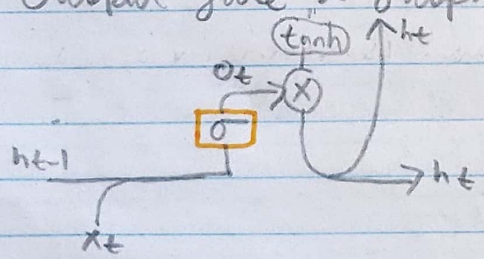
→ Input gate layer decides which values to update
→ tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to state

- Update old cell state to new cell state



$$\boxed{C_t = f_t * C_{t-1} + i_t * \tilde{C}_t}$$

- Output gate & output



$$\boxed{o_t = \sigma\left(W_o [h_{t-1}, x_t] + b_o\right)}$$

$$\boxed{h_t = o_t * \tanh(C_t)}$$

→ Sigmoid function to decide what parts are we gonna update
→ Put cell state through tanh & multiply by output of sigmoid gate so that we only output the parts we decided to

- Peepholes: Lets the gate layers look at cell state
→ Enables LSTM units to discover longer-range events in sequences

- For each input vector in the sequence of inputs, the error is equal to the sum of the deviations of all target signals from corresponding activations computed by the network

- LSTM training via BPTT is very computationally intensive for large number of steps (time steps)

e.g Input length 12 — Forward Pass: 12 time steps
Backward Pass: 12 time steps

- Lose info
- ↓ Computation cost

→ Truncated BPTT separates forward & backward passes into smaller operations

e.g Input length 12 — FP: 4 time steps x3 Parameter updates
BP: 4 time steps x3

Recursive Neural Networks

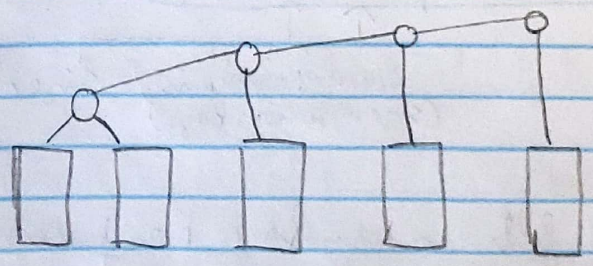
vs RNN

→ Models the hierarchical structures in the training set
e.g Deconstructing scenes in images, learning sequence & tree structures in NLP

- Shared weight matrix and binary tree structure

- Same set of weights applied recursively over a structured input by traversing a given structure in topology order

- Training - Backpropagation through structure (BPTS)



Each module maps 2 children to 1 parent

Chapter 6 - Deep Learning II

- Parameters: Properties of training data learnt during training
  - → e.g Weights, biases, word frequency, sentence length

- Hyperparameters: Cannot be learnt during training but are set beforehand
  - → Control parameters
  - → e.g Number of neurons, learning rate, loss function

- Number of layers & parameters count per layer determines how much capacity the network has to represent the data structure
  - → Type of layers can affect number of parameters in network

- FFNN Hidden layer count
  - → The larger the dataset, the more hidden layers can be used without overfitting
  - → Small network + large dataset $\xrightarrow{\text{(can)}}$ Underfitting

- FFNN Neuron count per layer
  - → Commonly but not always - neuron count of progressive layers declines
  - → Too many - increased chance of overfitting

- Calculating parameter count per network layer type

Fully connected	$n^{L-1} n^{L} + n^{L}$

Number of inputs (size of previous layer)

Size of current layer L

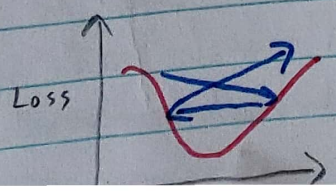
- Size of weights between any layer 1 and layer 2 is given by [nodes in layer 1 x nodes in layer 2]

- Activation function : Non-linear transformation done over the input signal
  - ReLU
    - → Piecewise linear function that will output the input directly if positive, zero otherwise
    - → Default activation function for many neural networks because a model that uses it is easier to train & often achieves better performance
    - → Overcomes vanishing gradient problem
  - Sigmoid
    - → Output layer for binary classification, values in range [0, 1]
  - Tanh
    - → Continuous data, more than [-1, 1]
    - → LSTM Layers
  - Softmax
    - → Multiclass classification model output layer
- Loss functions : Lets the optimization function know how well it is doing
  - → Output layers
  - → Layers that support unsupervised layer-wise pertaining

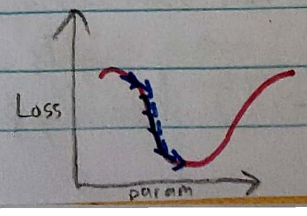
- Squared loss	Output layer	Regression
- Cross-entropy	Output layer	Binary classification
- Multi-class cross-entropy	Output layer	Classification
- Root MSE	Autoencoder, Output	Feature engineering, regression
- Negative log likelihood	Output layer	Classification

- Learning rate : Strong impact on stability & efficiency of training

Large Learning Rate : Unstable



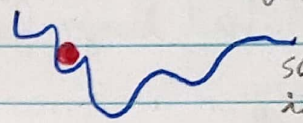
Small learning rate : Inefficient



- Learning rate schedule
  → Changes learning rate during learning, often between epochs
  → Time based vs Step based

- Adaptive learning rate
  → Adaptive GD algos such as Adagrad and Adam

- Momentum
  → Increases size of steps taken towards minimum trying to jump from local minima



SGD can get stuck in local minima

- Learning rate are network specific and depends on other parameters

Regularization : Process of introducing additional information to solve an ill-posed problem or to prevent overfitting
- 2 main categories
  → L1 & L2 : Reduce weights in parameter vectors that become too large
  → Dropout : Removing units/neurons

- Lasso Regression (L1)
  → Adds absolute value of magnitude of coefficients as penalty term to the loss function

$$\boxed{\lambda \sum_{j=1}^{p} |\beta_j|}$$

- Ridge Regression (L2)
  → Adds squared magnitude of coefficients as penalty term to the loss function

$$\boxed{\lambda \sum_{j=1}^{p} \beta_j^2}$$

→ L1 has less of a penalty for large weights but leads to many weights being driven to 0 → resulting vector can be sparse
→ L2 heavily penalizes large weights, but does not drive small weights to 0

→ L1 can be used for feature selection
→ L2 more often used

Early Stopping : Prevent Overfitting
- Stop training before weights have converged

- Dropouts
→ Works by setting a unit's activation to 0.0 temporarily
→ Randomly omitting neurons can prevent coadaptation and improve generalizat
→ Layers:
- Input layer - Probability between 0.5 and 1
- Hidden layer - Probability of 0.5
- Output layer - Usually no dropout

→ Often combined with other regularization methods
→ Issues:
- Less efficient as training samples rise up to tens of millions
- Increases training time (factor of 2 or 3)

Data Preparation : Need to transform data into a form in which tools can operate

Normalization : Transform data so it is more consistent
→ If inputs are too large, activations are too large
→ If inputs are too small, activations are too small
→ High valued attributes may overpower low valued ones

1) Standardization : Rescale feature so they have $\mu = 0$ $\sigma = 1$

$$\boxed{z = \frac{x - \mu}{\sigma}}$$

→ Gradient Descent methods benefits from this
→ Suitable for dense data, and when we expect outliers
→ Sparse data (lots of 0s in vectors) - often better to scale to [0,1]

2) Min-Max Scaling : Features are scaled to a fixed range [0,1]
→ More sensitive to outliers than standardization
→ Single outlier really affects min-max values

$$\boxed{x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}}$$

* Do normalization separately for training & test data!

- LSTM normalization
  → Consider all time-steps when calculating μ, σ, min or max

- CNN normalization
  → Share $\mu, \sigma, \ldots$ across all pixel locations, usually separated for each channel (RGB)

- Classification normalization
  → Normalize only inputs

- Regression normalization
  → Often need to normalize outputs
  → Must reverse normalization to predict original thing

Vectorization
→ A lot of considerations:
- What kind of source data we dealing with?
- What type of model we want to train with this data?
- What is our vectorization approach?

- Columnar Raw Data Attributes
  → Nominal: Finite set of possibilities. Enumerates, discrete, categorical
  → Ordinal: Nominal but with order

- Image Data (RGB for each pixel)
- Sequential Data
- Text Data

$$\boxed{TF\,IDF_{t,d} = tf_{t,d} \times idf_t}$$

Measures relative weight of how frequent the word is vs how distinct it is

Term Frequency

Inverse document frequency

$$tf_{t,d} = \frac{count(t)}{count(allTerms, Ind)}$$

$$idf_t = \log\left(\frac{N}{df_t}\right)$$

Total documents

# of documents that contain term t

Chapter 7 - Recommender Systems

- Recommender System: Utilize various sources of data to infer customer interests and suggest products, services, or something else
  - e.g Movie & book ratings, Buying a product, browsing products & websites

- Principles
  - Dependies exist between user-and item-centric activity
  - User interested in an action movie is more likely to be interested in another action movie rather than historic documentary
  - Collective buying or rating behavior of various users can be leveraged to create cohorts of similar users that are interested in similar products
  - These dependencies can be learned in a data-driven manner

- Categories
  - Collaborative filtering
    - → Use of ratings from multiple users in a collaborative way to predict missing ratings or make recommendations
  - Content-based Recommender Systems
    - → Discriptive attributes of items & user ratings are used to make recommendations
  - Knowledge-based Recommender Systems
    - → Users interactively specify their interests, and the user specification is combined with domain knowledge to provide recommendations
    - → Useful for items not purchased often and insufficient existing ratings
    - → Rating is performed on the basis of similarities between customer requirements and item descriptions

- Domain-specific recommender systems
  e.g Context-Based, Time-Sensitive, Location-Based, Social
  → 2 primary models
  1) Prediction version : Predict rating for user-item combination
  2) Ranking version : Top-K items for particular user recommendation

- Collaborative filtering
  - 2 types
  1) Memory-based collaborative filtering
     1) User-based
        → Like-minded users of a target user are used in order to make recommendations
     2) Item-based
        → Find items similar to target item that has been rated by user
  2) Model-based collaborative filtering
     → ML & data mining used as predictive models e.g Decision Trees

- User-based Memory/Neighbourhood models
  → Similarity function needs to be defined to identify similar users to target user and their similarity

$$\boxed{Sim(u,v) = Pearson(u,v) = \frac{\sum_K \in I_u \cap I_n (r_{uK} - \mu_u) \cdot (r_{vK} - \mu_v)}{\sqrt{\sum_K \in I_u \cap I_v (r_{uK} - \mu_u)^2} \cdot \sqrt{\sum_K \in I_u \cap I_v (r_{vK} - \mu_v)^2}}}$$

set of items users u & v specified rating

rating user u specified item K

user u mean rating

1) Pearson coefficient is computed between target user & all other users
2) Closest K users are found separately for each predicted item - the rated item!
3) Prediction-weighted average with the pearson coefficient of observed ratings in the top-K peer group

→ Problem? Different users use different scales.

user-user similarity between user 3 & other users

e.g

Item-Id / User-Id	1	2	3	4	5	6	Mean Rating	Pearson (i, 3) (user-user)
1	7	6	7	4	5	4	5.5	0.894
2	6	7	?	4	3	4	4.8	0.939
3	?	3	3	1	1	?	2	1.0
4	1	2	2	3	3	4	2.5	-1.0
5	1	?	1	2	3	3	2	-0.817

1) $$\text{Pearson}(1,3) = \frac{(6-5.5)(3-2)+(7-5.5)(3-2)+(4-5.5)(1-2)+(5-5.5)(1-2)}{\sqrt{(3-2)^2+(3-2)^2+(1-2)^2+(1-2)^2}\cdot\sqrt{(6-5.5)^2+(7-5.5)^2+(4-5.5)^2+(5-5.5)^2}}$$

$= 0.894$ Do it for the rest

2) 2 most similar users: 1 and 2 to user 3

3) Weighted average prediction $\hat{r}_{31} = \frac{7\times0.894 + 6\times0.939}{0.894+0.939} = 6.49$

$$\hat{r}_{36} = \frac{4\times0.894 + 4\times0.939}{0.894+0.939} = 4$$

- Issue? Overly optimistic estimates & different users use different scales
  → Ratings need to be mean-centered in row-wise fashion
  → Use those mean-centred values in remaining calculation

Item-Id / User-Id	1	2	3	4	5	6
1	1.5	0.5	1.5	-1.5	-0.5	-1.5
2	1.2	2.2	?	-0.8	-1.8	-0.8
3	?	1	1	-1	-1	?
4	-1.5	-0.5	-0.5	0.5	0.5	1.5
5	-1	?	1	0	1	1

$$\hat{r}_{31} = \underbrace{2}_{\text{user 3 mean}} + \frac{1.5 * 0.894 + 1.2 * 0.939}{0.894+0.939} = 3.35 \qquad \hat{r}_{36} = 0.86$$

- Can use other similarity measures (such as cosine)
- Can use different ways of centering (Z-score)
- Can use variants of the prediction function.
- Can use different ways of selecting peer groups

Item-based Memory/Neighbourhood Models

→ Similarities computed between items
→ Leverage user's own ratings on similar items
→ Peer groups are constructed using items

mean centred rating

$$\text{Adjusted Cosine}(i,j) = \frac{\sum_{u \in U_i \cap U_j} s_{ui} \cdot s_{uj}}{\sqrt{\sum_{u \in U_i \cap U_j} s_{ui}^2} \cdot \sqrt{\sum_{u \in U_i \cap U_j} s_{uj}^2}}$$

users that specified ratings for item i

Mean Centric

Item-Id	1	2	3	4	5	6
User-Id						
1	1.5	0.5	1.5	-1.5	-0.5	-1.5
2	1.2	2.2	?	-0.8	-1.8	-0.8
3	?	1	1	-1	-1	?
4	-1.5	-0.5	-0.5	-0.5	0.5	1.5
5	-1	?	-1	0	1	1
Cosine (1,j)	1	0.735	0.912	-0.848	-0.813	-0.990
Cosine (6,j)	-0.990	-0.622	-0.912	0.829	0.730	1

(item-item)

user 3 ←

1) $$\text{Adjusted Cosine}(1,3) = \frac{1.5 * 1.5 + (-1.5 * -0.5) + (-1 * -1)}{\sqrt{1.5^2 + (-1.5)^2 + (-1)^2} \cdot \sqrt{1.5^2 + (-0.5)^2 + (-1)^2}}$$

$$= 0.912$$

Do it for rest

2) • Items 2 and 3 most similar to 1
• Items 4 and 5 most similar to 6

3) $\hat{r}_{31} = 2 + \frac{1 * 0.735 + 1 * 0.912}{0.735 + 0.912} = 3$

$\hat{r}_{36} = 2 + \frac{-1 * 0.735 - 1 * 0.912}{0.735 + 0.912} = 1$

- Memory-based advantages & disadvantages
- Advantages
  - Simple & intuitive
  - Can justify recommendations
  - Recommendations relatively stable with additions of new items & users
- Disadvantages
  - Similarity calculation may take long in large-scale setting
  - Limited coverage because of sparsity

- Model-based collaborative filtering
  → Summarized model of data created upfront, as with supervised & unsupervised machine learning methods
  → Training clearly separated from prediction phase
  → e.g SVMs, NNs

- Using an Arbitrary Classification model as Black-Box
  → A classification algorithm is used as a black-box to predict ratings of one of the items with the ratings of other items
  → Iteratively fill in missing values of training columns with successive refinements

1) Initialize missing entries in the matrix with row or column averages
2) Two step iterative approach
   a) Use algo A to estimate missing entries of each column by setting it as target variable & remaining columns as feature variables. Observed ratings in the target column are used for training, and missing ratings are predicted

b) Update all missing entries based on prediction of algo A on each target column

	I1	I2	I3	I4	
u1	5	4	avg^c	predict	Test
u2	avg^c	3	4	predict	
u3	avg^c	2	5	3	Train
	Features			Target	

- Content-based recommender systems
  - → Discriptive attributes of items & user's own rating on other movies are sufficient to discover meaningful recommendations
  - → Particularly useful when item is new
  - → Well suited to giving recommendations in text-rich & unstructured domains

- Components:
  - Preprocessing and feature extraction
  - Content-based learning of user profiles
  - Filtering and recommendations

- Preprocessing and feature extraction
  - → To account for common words - IDF
  - → Feature selection & weighting - Gini Index

$$\boxed{Gini(w) = 1 - \sum_{i=1}^{t} p_i(w)^2}$$

t - total number of possible values of the rating

$p_i(w)$ - fraction $\frac{\text{\# of items with w rated at 1 (or each of t values)}}{\text{\# of items with w}}$

Prob of object being classified to particular class

→ Measures degree or probability of a particular variable being wrongly classified when it is randomly chosen

- Learning user profiles & filtering
  - → Possible approach nearest-neighbor classification
    - → Same as user/item based neighbour models technique via similarity measures

- IDF & TDIDF example

→ Document, 100 words, "Cat" appears 3 times $tf = \frac{3}{100}$

→ 10 million documents, "Cat" appears in 1000 $idf = \log\left(\frac{10,000,000}{1000}\right)$

→ $tfidf = 0.03 * 4 = 0.12$ //

- Gini example

→ P(Past Trend = Positive) = 6/10
→ P(Past Trend = Negative) = 4/10
→ If (PT = Positive, Return = Up), P = 4/6
→ If (PT = Positive, Return = Down), P = 2/6
→ Gini Index $= 1 - \left(\left(\frac{4}{6}\right)^2 + \left(\frac{2}{6}\right)^2\right) = 0.45$

→ If (PT = Negative, Return = Up), P = 0
→ If (PT = Negative, Return = Down), P = 4/4
→ Gini Index $= 1 - \left((0)^2 + \left(\frac{4}{4}\right)^2\right) = 0$

→ Gini Index for PT $= \left(\frac{6}{10} * 0.45\right) + \left(\frac{4}{10} * 0\right) = 0.27$

- Evaluating Recommender systems
  - → Similar to evaluating classification & regression models
    - Online: Click-through rate, conversion rate
    - Offline: Evaluations with historical data

- Designing accuracy evaluations
  → e.g hold-out & cross-validation

- Accuracy metrics:
  → Accuracy of estimating ratings
  → Accuracy of estimating rankings

- Accuracy of estimating ratings
  → Entry specific error : $\boxed{e_{u;} = \hat{r}_{u;} - r_{u;}}$
  → Accuracy: MSE, MAE, RMSE

$$\boxed{MSE = \frac{\sum_{(u,i) \in E} e_{u;}^2}{|E|}}$$

$$\boxed{MAE = \frac{\sum_{(u,i) \in E} |e_{u;}|}{|E|}}$$

$$\boxed{RMSE = \sqrt{MSE}}$$

- Accuracy of estimating rankings
  → Can use Spearman correlation coefficient

* Q: You want to predict user u rating for a new item i (nobody rated item i yet) What approach would you use?

A. Item-based filtering
B. User-based filtering
C. Content-based recommender System

C

Chapter 8 - Big Data

- Big Data Challenges:

1) Volume [PCCC - VNFB]

- Processing Performance
  - → Volume adds computation complexity
  - → Standard SVM has training time complexity $O(m^3)$ Space complexity $O(m^2)$
  - → Time often increases exponentially

- Curse of Modularity
  - → Many learning algorithms rely on the assumption that the data being processed can be held entirely in memory or in a single file on a disk
  - → MapReduce - scalable programming paradigm for processing large datasets by means of parallel execution on a large number of nodes

- Class Imbalance
  - → As datasets grow larger, the assumption that the data are uniformly distributed across all classes is often broken
  - → Algorithms are negatively affected by class imbalance

- Curse of dimensionality
  - → Many features = slow
  - → Difficulties encountered when working in high dimensional space (high number of features or attributes)
  - → Predictive ability and effectiveness of an algorithm decreases as dimensionality increases for a training set of static size
  - → Greater the data, greater potential for high dimensionality

- Feature Engineering & Selection
  - → More difficult when data grows horizontally & vertically

- Variance & bias [Generalization errors]
  → Variance: Consistency of a learner's ability to predict things
  → Bias: Learner's ability to learn the wrong thing

- Non-linearity
  → Correlation coefficient only fully meaningful if linear relationship exists between variables

- Bonferonni's principle
  → An algorithm that we think is useful for finding a particular set of data actually returns more false positives as it returns larger portions of data than should be within that category. [Remember cheating spouse example]

2) Variety [DDD]

- Data Heterogenity
  → Syntactic heterogenity: Diversity in data types & file formats
  → Semantic heterogenity: Differences in meanings & interpretations
  → ML approaches were not developed to handle semantically diverse data

- Data Locality
  → Big data usually distributed over large number of files residing in different physical locations
  → ML algorithms assume that entire dataset is found in memory or in a single disk file

- Dirty & noisy data
  → Data contain various types of measurement errors, outliers, and missing values
  → Big Data typically described as ill-conditioned due to the amount of time & resources necessary to get them ready for analysis

3) Velocity [DDD-IC]

- Data Availability
  - → ML approaches assume entire dataset present before learning began
  - → Streaming - data arriving constantly
  - → Incremental learning (sequential learning) - Algorithm's ability to adapt its learning based on the arrival of new data without the need to retrain on the complete dataset

- Concept Drift
  - → Changes in the conditional distribution of target output given the input, while distribution of input itself may remain unchanged
  - → Problem occurs when machine learning models are built using older data that no longer accurately reflect distribution of new data

- Independent & Identically Distributed Random Variables (i.d.d)
  - → i.i.d requires data to be in random order while many datasets have a pre-existing non-random order
  - → Not realistic to randomize with Big Data

- Data Provenance
  - → Process of tracing and recording origin of data & their movements between locations
  - → Helps with data quality
  - → With Big Data, the provenance dataset itself becomes too large

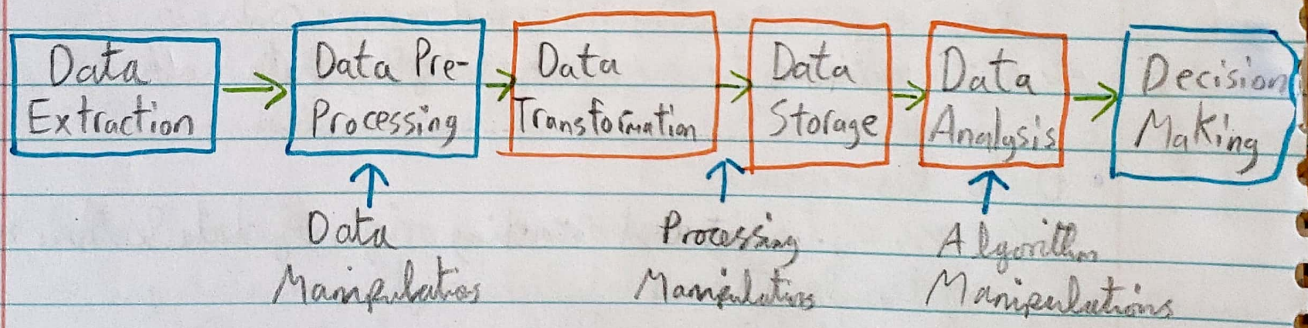
- Data Uncertainty
  - → Means & methods used to gather data can introduce uncertainty
  - → Lack of objectivity
  - → e.g social media data for sentiment analysis

- Approaches for ML to deal with Big Data:
  1) Manipulations
     - Data manipulations
     - Processing manipulations
     - Algorithm manipulations
  2) Learning paradigms

- Cloud based solutions

Advantages	Disadvantages
• Flexibility	• Network connection dependency
• Cost-efficiency	• Limited features
• Ease of use	• Loss of control
• Backup and recovery	• Security
• Off-site access	• Technical issues
[CE-FOB]	[LL-TSN]

- Data Analytics Pipeline



- Data Manipulation
  - Dimensionality reduction - without loss of information
  - Data cleaning
  - Instance selection - selecting data subset that represents whole dataset

- Processing Manipulations - modifying how data is processed and stored
  - Vertical scaling (scale up) - increasing hardware capabilities
  - Horizontal scaling (scale out) - distributed systems processing distributed over
    - Batch-oriented: - Processing large amount of data at once
      - More concerned with throughput than latency
      - Map-reduce based solutions such as hadoop

- Stream-oriented: - Operate on small set of data in (near) real-time
  - Apache Storm, Spark Streaming

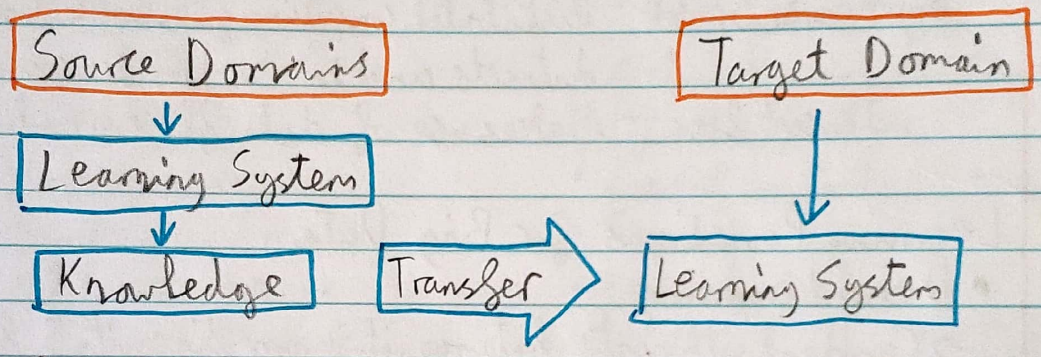
- Algorithm Manipulations - modify existing algorithms with or without new paradigms
  - Algorithm modification - Pegasos provides an optimized SVM
  - Algorithm modification with new paradigms
    → Modifying algorithms to work better with new process manipulations and or new paradigms
    e.g Apache Spark - Distributed computing framework based on distributed datasets and in-memory processing
    TensorFlow - Makes use of data flow graphs

- Learning Paradigms for Big Data
  - Deep Learning
    → Can deal with data heterogeneity & non-linearity (& feature engineering as there is no need for it)
    → Can deal with complex problems
    → Cannot do this in computationally efficient way

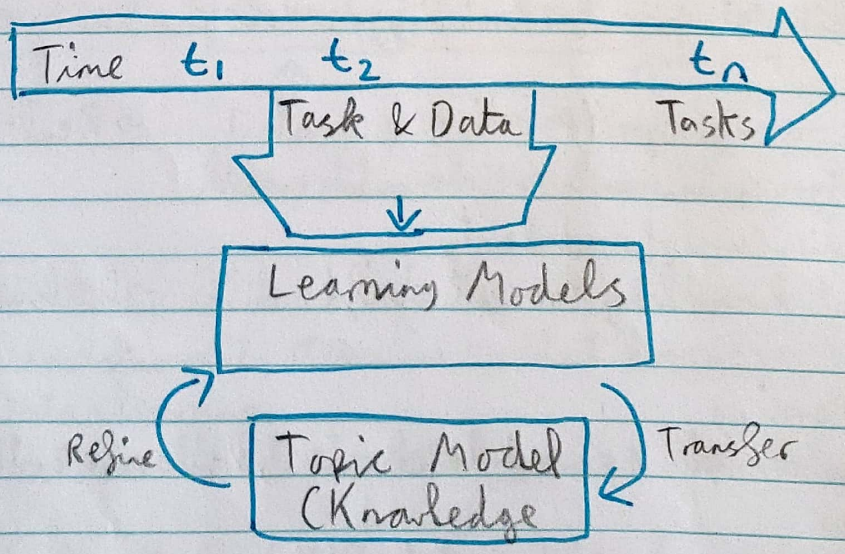
  - Online Learning
    → Uses data streams for training, models can learn one instance at a time
    → "Learn-as-you-go" paradigm
    → Real-time processing
    → Paradigm continuously maintains its model
    → Remedies curse of modularity
    → Good at handling concept drift
    → Deals with i.i.d issue well

  - Local Learning
    → Separate input space into clusters, build a separate model for each cluster
    → Often much more efficient to find a solution for k problems of size m/k than for a single problem of size m
    → Remedies curse of modularity
    → Reduces challenges caused by class imbalances and data locality
    → Often can improve performance
    → Deals with variance & bias issues well

- Transfer Learning
  - → Improve learning in a particular domain (target domain) by training model with other datasets from multiple domains (source domains) with similar attributes or features such as problem and constraints
  - → Different elements transfered: Instances, feature representations, model parameters, and relational knowledge
  - → Issues? Data heterogeneity & class imbalance



- Lifelong Learning
  - → Learning is continuous; knowledge is retained & used to solve different problems
  - → Related to online & transfer learning



- Ensemble Learning
  - → Combines multiple learners to obtain better learning outcomes than those obtained from any constituent learner
  - → 2 ways
    - Train different learners, each one on the complete dataset
    - Train each learner (same or different) only on a subset
  - → Outcomes from all learners are combined using some kind of voting scheme to improve accuracy of overall solution
  - → Can be used to improve accuracy and/or computation time

* Distributed computing can assist with dealing with processing performance when ML algorithm can be efficiently parallelized and data transfer between compute nodes is not large