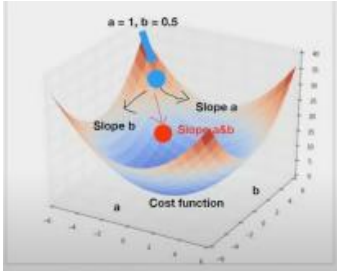
DES423 Lecture Summary

Applied Machine Learning and Artificial Intelligence (Final Exam)
SIIT DE-ASD Y4T1/2022 – By Paphana Yiwsiw (@waterthatfrozen)

Contents

Gradient Descent

- "A gradient descent measure how much the function output changes if you change input a little"
- Used to find linear regression model (y=ax+b) in machine learning.
- The difference between y(actual value) and h(predicted value) is called an error. Smaller, better.
- **Loss function** is used to calculate for an error. Square error is widely used.
- **Sum of error = cost function:**
  - Cost function or Sum of Square Error: SSE
    $$\sum_{i=1}^{n} \frac{1}{2}(y_i - h_i)^2$$
  - Slope of axis a
    $$-(y_i - h_i)x$$
  - Slope of axis b
    $$-(y_i - h_i)$$
- Obtaining the minimum point by gradient descent
  - 

  - First, guess **a** and **b** in y=ax+b
  - Calculate Cost function (SSE)
  - Find slope along axis a and b
  - Adjust new a and b (α is learning rate, should be very small)
    $$a = a - \alpha\frac{\partial(\text{SSE})}{\partial(a)} \qquad b = b - \alpha\frac{\partial(\text{SSE})}{\partial(b)}$$
  - Repeat until we cannot get the lower cost or cost ›› 0

Give: $a=1,\ b=0.5$ h=ax+b $\alpha = 0.01$

x (Input)	y (Output)	h (Predict)	Error	-(y-h)	-(y-h)x
0	1	0.5	0.125	-0.5	0
1	3	1.5	1.125	-1.5	-1.5
2	5	2.5	3.125	-2.5	-5.0
3	7	3.5	6.125	-3.5	-10.5
		Cost (Sum)	10.5	-8	-17

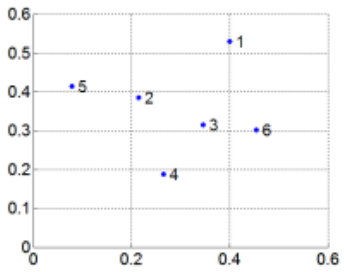
$$\frac{\partial(SSE)}{\partial(a)} = -(y_i - h_i)x = -17 \rightarrow a = a - \alpha\frac{\partial(SSE)}{\partial(a)} = 1 - (0.01)(-17) = 1.17$$

$$\frac{\partial(SSE)}{\partial(b)} = -(y_i - h_i) = -8 \rightarrow b = b - \alpha\frac{\partial(SSE)}{\partial(b)} = 0.5 - (0.01)(-8) = 0.58$$

Clustering

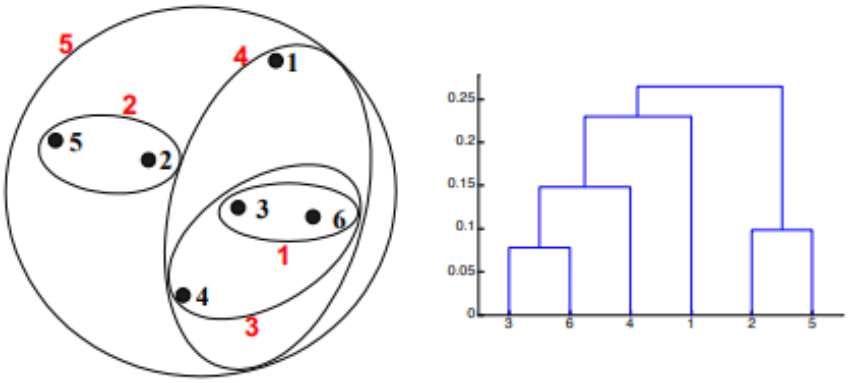
- Unsupervised learning method, determining grouping among unlabeled data.
- Task of grouping objects in such a way that objects in the cluster are more like each other in the cluster than those outside the cluster.

Clustering Algorithms

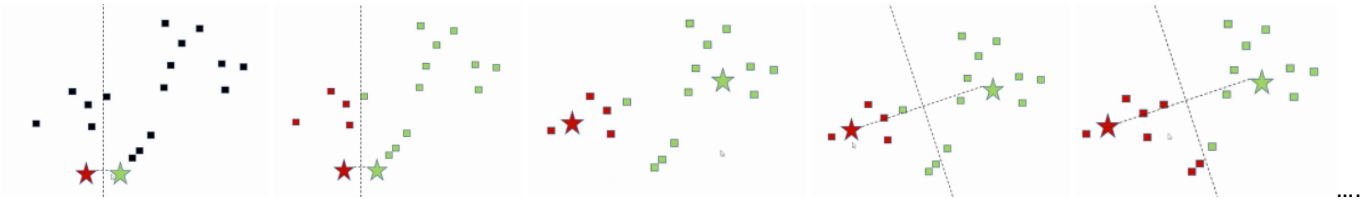
- **Hierarchical clustering**
  - Produces a set of nested clusters organized as a hierarchical tree, view as dendrogram.
  - Any preferred number of clusters is obtained by cutting the dendrogram at that level.
  - **Two main types of algorithms**
    - **BOTTOM-UP: Agglomerative Clustering Algorithm**
      - Start with each point as individual clusters, each step merges the closest pair until only one or k clusters left.
    - **TOP-DOWN: Divisive**
      - Start with all-inclusive one cluster, each step splits a cluster until each cluster contains 1 point or there are K clusters.
    - Normally use similarity or distance matrix to merge or split one cluster at a time.
  - **Cluster distance measures**
    - MIN/Single-Link: closest element in clusters.
      SENSITIVE TO OUTLIER
    - MAX/Complete-Link: farthest element in clusters.
      BIASED TOWARD GLOBAL CLUSTER/TEND TO BREAK LARGE CLUSTER
    - GROUP AVERAGE/Average-Link: average of all pairwise distances
      Compromise between MIN and MAX, still biased toward global cluster
    - DIST BW CENTROID/Centroids: distance between centroids of clusters
    - WARD'S: how does it change the total distance from centroids if joined.
  - Proximity Matrix
    - 

Distance Matrix:

	p1	p2	p3	p4	p5	p6
p1	0.00	0.24	0.22	0.37	0.34	0.23
p2	0.24	0.00	0.15	0.20	0.14	0.25
p3	0.22	0.15	0.00	0.15	0.28	0.11
p4	0.37	0.20	0.15	0.00	0.29	0.22
p5	0.34	0.14	0.28	0.29	0.00	0.39
p6	0.23	0.25	0.11	0.22	0.39	0.00

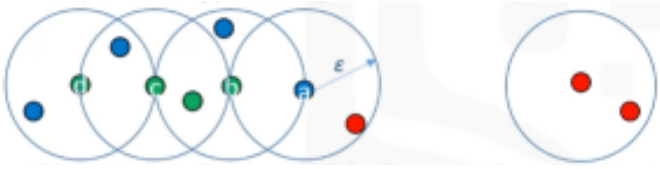
  - Dendrogram
    - Different dendrogram may caused by different distance measurements
    - 

- K-means clustering
    - Partitional clustering approach, specify number of cluster (K)
    - Each cluster is associated with a centroid, each point assigned to closed centroid.
    - Closeness measured by Euclidean distance
    - Algorithm steps
        - Select K points as initial centroid
        - Repeat[Form K clusters by assign to the closest centroid, calculate new centroid] until all centroids don't change.



    - Most convergence happens in first few iterations; initial centroids chose randomly. Centroids are normally mean of points in the cluster. Complexity O(n*K*I*D)
- Density-based clustering
    - Core, Border, Outlier
      Core point: has min number of points within specified radius (ε), including itself.

    - **Density reachability**
        - Y is reachable from X
          if there is a path that starts with X and ends with Y where each point in between must be core point.
        - Y is directly density-reachable from X,
          if X is a core object and Y is in X ε-neighborhood.



        - A is directly reachable from B, B is directly reachable from C.
        - A is *indirectly* reachable from D
        - D is not density-reachable from A since A is not a core point.
- How does it work?
    - Pick a random point that is not assigned to cluster or outlier, determine if it is a core point. If not, label as outlier.
    - Once core point has been found, add all directly reachable to its clusters. Jump to those points, add to the cluster. If outlier added, label as border point.
    - Repeat until every point are assigned to cluster or outlier point.

Association Rule Mining

- Important ML concept used in market basket analysis through customer transaction to understand customer behavior for marketing purposes, management, and CRM.
- Discovering patterns and association hidden in large datasets.

Frequent Itemset

- **I** is the set of items in the store, **D** is the set of transaction which contains item in I (itemset)
- There are [2^(number of items)]-1 subsets. E.g., 5 items = 31 possible itemset.
- Itemset of size K = itemset contain K items.
- **Support:** frequency of an itemset is in the number of transactions that contains X.

Transaction	Item appearing in the transaction
T1	{pasta, lemon, bread, orange}
T2	{pasta, lemon}
T3	{pasta, orange, cake}
T4	{pasta, lemon, orange, cake}

  - sup(X) = sup({pasta, orange}) = 3 or 0.75 (ratio or absolute support)

Association Rule Mining Tasks

FREQUENT ITEMSET MINING/GENERATION

- There are minsup, set by user; minimum support value to call that itemset as frequent.
- Support of that itemset must be greater or equal to minsup.
- **NAÏVE APPROACH**
  - Count the support of all itemset, read each transaction in the database and count all, waste of performance, too many comparisons.
- **APRIORI ALGORITHM**
  - **Property 1:** Let there be two itemset X and Y. If X is subset of Y, the support of Y is less than or equal to support of X. (Support of Y cannot be greater than support of X)
  - **Property 2:** Let there be an itemset Y. If there exists an itemset X is subset of Y such that X is infrequent, then Y is infrequent. (X is infrequent then Y is infrequent)
  - **Steps in APRIORI algorithm**
    1. Scan DB. Calculate the support of all itemset of size 1
    2. Eliminate infrequent itemsets.
    3. Generate candidate of size 2 by pair of frequent itemsets of size 1.
    4. Eliminate candidates of size 2 that have **infrequent subset** according to property 2.
    5. Scan DB. Calculate the support of candidate itemsets of size 2
    6. Eliminate infrequent itemsets of size 2
    7. Generate candidate of size 3 by pair of frequent itemsets of size 2
    8. Eliminate candidates of size 3 that have **infrequent subset** according to property
    9. ***Repeat (Scan, calculate support, eliminate, generate, eliminate subsets, …) until no frequent itemset is left.***

RULES GENERATION

- After obtaining frequent itemset of each size from APRIORI process, convert to absolute support
- Association rule of A→B: there is an association between occurrence of itemset A (**LHS**/antecedent) and itemset B (**RHS/**consequent).
- Noted that LHS→RHS not equal to RHS→LHS

- Measurement of association's strength are confidence and lift.
- **CONFIDENCE:** likeliness of occurrence of RHS on the cart given the cart already have LHS.

$$\text{Confidence}(\text{LHS} \rightarrow \text{RHS}) = \frac{\text{support}(\text{LHS} \cap \text{RHS})}{\text{support}(\text{LHS})}$$

  - It doesn't matter what you have in LHS for such frequent RHS. The confidence will always be high if you have a very frequent RHS.

In this case Confidence(Toothbrush→Milk)
will always be high regardless of their weak association.

- **LIFT:** rise in probability of occurrence of RHS on the cart with knowledge of LHS being present **over** probability of having RHS on the car without any knowledge about presence of LHS.

$$\text{Lift}(\text{LHS} \rightarrow \text{RHS}) = \frac{\text{Confidence}(\text{LHS} \rightarrow \text{RHS})}{\text{support}(\text{RHS})}$$

  - **Lift > 1:** LHS actually leads to RHS on the cart
  - **Lift = 1:** LHS and RHS are independent
  - **Lift < 1:** LHS **doesn't lead to** RHS on the cart.
  -

More the value of lift, greater are the chances of preference to buy Product-**RHS** if the customer has already bought Product-**LHS**. Lift is the measure that will help store managers to decide product placements on aisle.

Transaction	Item appearing in the transaction
T1	{pasta, lemon, bread, orange}
T2	{pasta, lemon}
T3	{pasta, orange, cake}
T4	{pasta, lemon, orange, cake}

Rule	Support(RHS)	Confidence	Lift
{lemon => orange}	0.75	0.67	0.67/0.75=0.89
{orange => lemon}	0.75	0.67	0.67/0.75=0.89
{orange => cake}	0.50	0.67	0.67/0.50=1.34
{cake => orange}	0.75	1.00	1.00/0.75=1.33

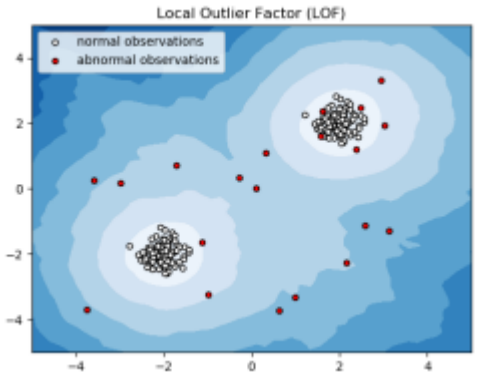
Anomaly Detection

- An anomaly is a deviation or exception that differs from a norm. Most likely a suspicious activity that doesn't fit into the pattern of events.
- One of the most common use cases in ML, helps to identify and prevent frauds.
- Mostly used for intrusion detection, fraud detection, health monitoring, defect detection.
- Type of Anomaly
    - Global Outliers

        - A data point assumes a value that is far outside all other data points value range can be considered as global anomaly**.** Rare event.
    - Contextual Outliers

        - A data point value doesn't correspond with what we expect from similar pattern and observation in the same context, usually temporal (over time).

LOF: Local Outlier Factor Algorithm



- The most common technique for anomaly detection, based on local density concept.
- Compare local density of an object with that of neighboring data points.
- If a data point has a lower density than its neighbors, then it is considered as an outlier.

Isolation Forest

Get the anomaly point using similar algorithm to random forest.

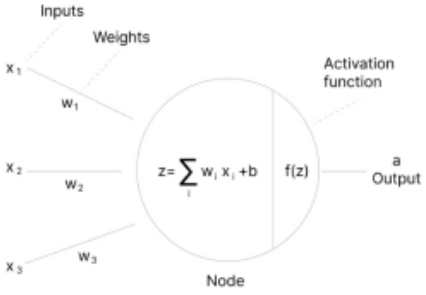
ANN: Artificial Neural Networks

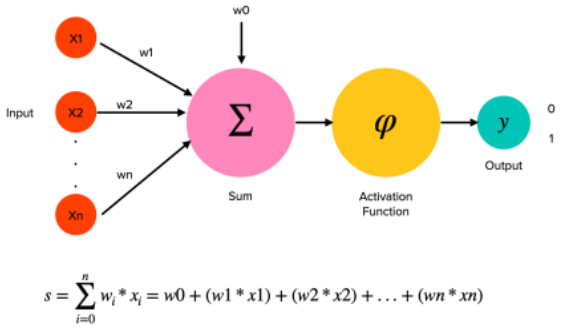
Classification Problem in ANN

- Data can be separated by a line, but how are we going to find this line to create a classifier.
- Line: linear boundary line: Wx+b=0 where W=(w1,w2, …), x=(x1,x2, …), b=bias.
    - If Wx+b >= 0, then predict as 1, otherwise 0.

Perceptron

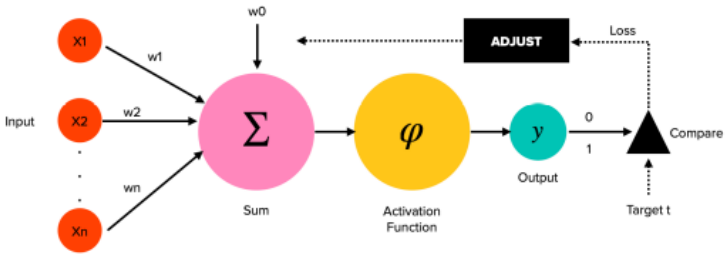
- Building blocks of ANN: set of nodes combined to solve problem, converting element into a graph
- Elements in perceptron: Input, weights, bias. Output: accept when > 0 otherwise reject.

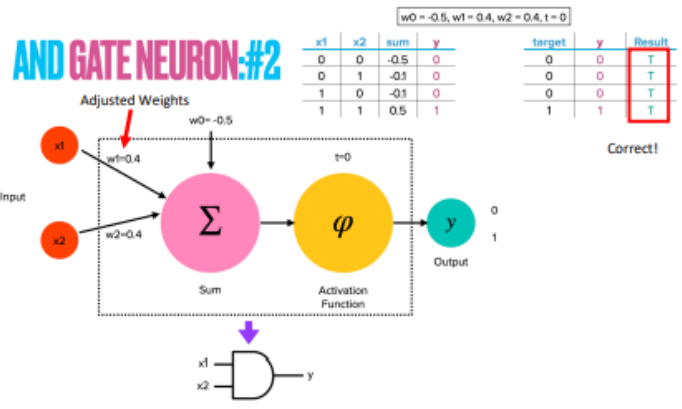




$$s = \sum_{i=0}^{n} w_i * x_i = w0 + (w1 * x1) + (w2 * x2) + \ldots + (wn * xn)$$

- **Activation Functions**
    - **Binary Step:** threshold-based activation function, activate when >= threshold
    - **Linear:** a.k.a. no activation or identity function, activation is proportional to the input of the function, simply put that input out.
    - **Sigmoid:** take any real value, output = [0,1]. More positive, closer to 1. f(x) = 1/(1+e^(-x))
    - **Tanh:** output centered at 0, strongly negative or positive. Output = [-1,1].
      f(x) = (e^(x) – e^(-x)) / (e^(x) + e^(-x))
    - **RELU:** rectified linear unit. Most common in hidden layer, deactivate when output of linear transformation is less than 0; f(x) = max(0, x)
    - **Softmax:** combination of many sigmoids. Used in last layer of multiclass classification.
- Back Propagation





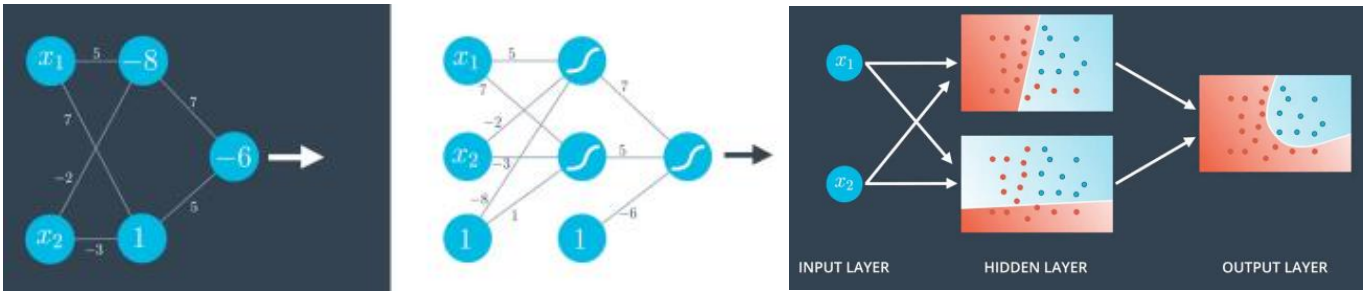
- The problem with XOR gate is that it can't be done with single line/perceptron.

Perceptron Algorithm

- Algorithm builds model themselves, neural networks doesn't know where to start, so we generate random weights for linear equation, try to move a line around until it gets better, moving too drastically to the points might cause other to be misclassified.
- **Learning Rate**
    - Tuning parameter in an optimization algorithm, determine the step size at each iteration while moving toward a minimum of a loss function.
    - A small number multiplying with current weights to make a small move towards the point. Used repeatedly in perceptron until it reaches its goal.
    - Step to adjust weights of perceptron
        1. Start with random weights (w_1, …, w_n) and bias (b)
        2. For every misclassified point (x_1, …, x_n)
            a. If a prediction = 0 (actual = 1): change b to b + α, w_i to $w_i + \alpha x_i$
            b. If a prediction = 1 (actual = 0): change b to b – α, w_i to $w_i - \alpha x_i$

Non-linear regions

- Perceptron algorithm only works with a linear line. For more complex, use neural networks.
- Error function or loss function tells us how far we are from the solution, neural network adjusts the model themselves repeatedly until reached goal.
    - Gradient descent problem used to fund the right direction to move towards solution.
        - Global minima problem → Stochastic Gradient Descent. It used sum of error distance instead, small variation/continuous is better. Check error every time you make a move, keep moving until reach the goal.
        - To calculate the continuous prediction, change the activation function. E.g., step function for binary result, sigmoid function returns a range of 0 to 1.
        - To calculate error, use average. Calculate error use a "minus logarithm"
        - Cross Entropy: used to determine which model is better. Smaller, the better.
        - What if its multiclass? Use One-hot encoding or SoftMax function.
    - Probability space of non-linear region
        - Combine two lines into non-linear model and add using sigmoid function. Weight and bias are given to each model.

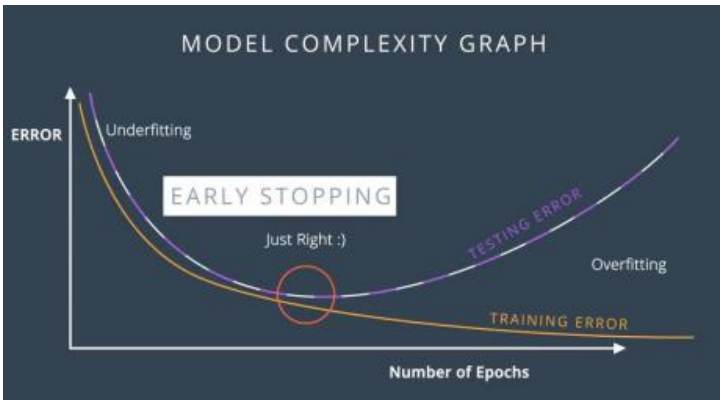


    - Layer in neural network architecture
        - First layer: input layer
        - Between first and last layer: hidden layer, takes in set of weighted input and produce output through activation function
        - Last layer: output layer: combine all linear model to obtain nonlinear model
        - More layers = deep neural network, more complex model, usually in multiclass.

Optimization of ANN Architecture

- Gradient descent → Get to the bottom. Learning Rate → take tiny steps, solve underfitting
- Algorithms: Feedforward, backpropagation.
- Error function tells how bad predictions are, measure distance between target and predicted value. Error should have no direction, use squared error instead. "Sum of squared error" or SSE used to find gradient for weight updates in the backpropagation.
    - New weight = current weight + change in gradient and apply learning rate.
- **Underfitting:** Model can neither model the training data nor generalized to new data
- **Overfitting:** Model that model according to training data too well.

Early Stopping



Regularization

- Technique used to calibrate ML models to minimize the adjusted loss function and prevent overfitting or underfitting.

$$\textbf{L1 ERROR FUNCTION} = -\frac{1}{m}\sum_{i=1}^{m}(1-y_i)ln(1-\hat{y}_i)+y_i ln(\hat{y}_i)+\boxed{\lambda(|w_1|+\ldots+|w_n|)}$$

$$\textbf{L2 ERROR FUNCTION} = -\frac{1}{m}\sum_{i=1}^{m}(1-y_i)ln(1-\hat{y}_i)+y_i ln(\hat{y}_i)+\boxed{\lambda(w_1{}^2+\ldots+w_n{}^2)}$$

-
- **L1 Regularization/Lasso:** modify overfitted/underfitted model by adding penalty equivalent to the SUM OF ABSOLUTE VALUES OF COEFFICIENTS.
    - Good for feature selection, small weights tend to go to zero. L1 helps to select which features are important, what is not important will turn to zero.
- **L2 Regularization/Ridge:** modify overfitted/underfitted model by adding penalty equivalent to the SUM OF SQUARES OF THE MAGNITUDE OF COEFFICIENTS
    - Better for training model, maintain weight homogenously small, better results.
- Lambda tells how much we want to penalize the coefficients, larger the lambda, more penalize.

Dropout

- Turn off some nodes randomly in each epoch, based on probability.

Gradient Descent

- Local minima problem → Solution: random restart.
- **Vanishing Gradient**
    - **Small gradient:** weights and bias of initial layers will not be effectively updated with each training sessions, worst in back propagation.
    - Gradient exponentially decreases as propagated down to initial layers. May stuck at local minima because of tiny steps.

- **Solution: change activation function.**
  - Tanh: for deep learning algorithms
  - ReLU: use a lot in sigmoid, improve training without losing accuracy.
  - For multilayer perceptron, use multiple Relu units.

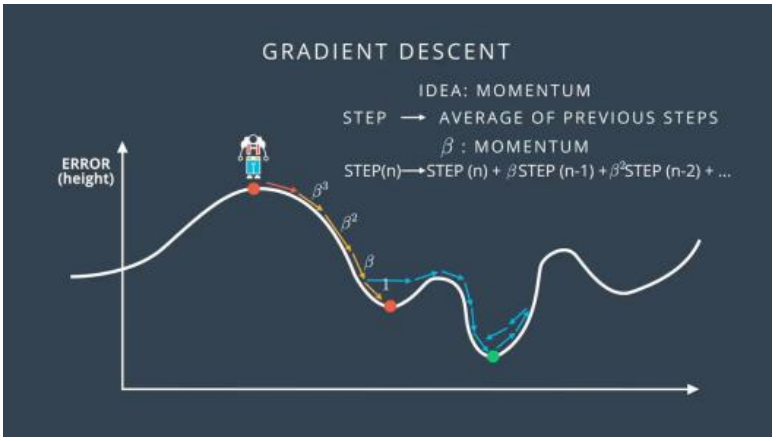
Stochastic Gradient Descent

- Take small subset of data (one batch), run them through neural network (feedforward, error calculation, update weights, backpropagation), calculate gradient of error function, then move one step in that direction until no batch is available.

Learning Rate Decay

- Too big step, miss the minimum and keep the model chaotic.
- Too small step, higher chance of local minima but slow training
- Model not working? Reduce learning rate.

Momentum



- Configuration: number of previous steps is important, weight each step that matter a lot, what not matter, weight less. This will get us over the hump and to global minima.

Error functions

- Classification Loss Functions
  - Binary: Binary cross entropy, Hinge loss
  - Multiclass: Categorical cross entropy loss, Kullback Leibler divergence loss
- Regression Loss Functions
  - SSE, MSE, RMSE, Mean Squared Logarithmic Error Loss, MAE