# CSCI 4587/5587, Fall 2023
# Machine Learning I

# Study Guide for Final Exam

**Date/Time of Final Exam: Tuesday, Dec/05/2023, 03:00 PM to 5:00 PM**

**(Please don't distribute this study guide.
The guide is for your study purpose only)**

**1. How would you briefly define Machine Learning in contrast to the traditional programming model?**

**Ans**: Machine Learning (ML) is about building systems that can learn from data. Learning means getting better at some task, given some performance measures. Eventually, ML gives computers the ability to learn without being explicitly programmed.

Traditional programming models versus the ML models are compared using the figures below:

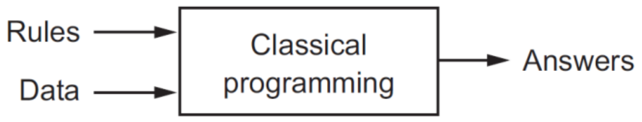


**Fig. 1**: Traditional programming Model.

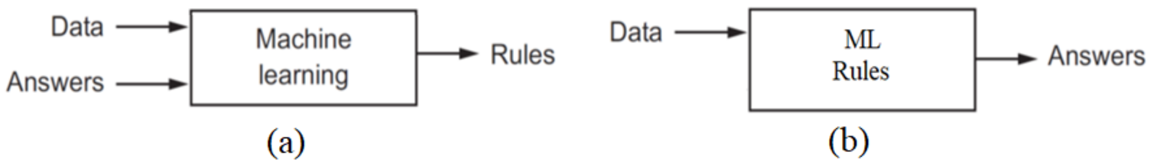


**Fig. 2**: Machine learning: a new and powerful programming paradigm.

**02. (a) What is a labeled training set? (b) What are the two most common supervised tasks? (c) Can you name four common unsupervised tasks?**

**Ans**: (a) A labeled training set is a training set that contains the desired solution (a.k.a. a label) for each instance. (b) The two most common supervised tasks are regression and classification. (c) Common unsupervised tasks include clustering, visualization, dimensionality reduction, and association rule learning.

**03. (a) Would you frame the problem of spam detection as a supervised learning problem or an unsupervised learning problem? (b) What is an online learning system? (c) What is out-of-core learning?**

**Ans**: (a) Spam detection is a typical supervised learning problem: the algorithm is fed many emails along with their labels (spam or not spam). (b) An online learning system can learn incrementally, as opposed to a batch learning system. This makes it capable of adapting rapidly to changing data and autonomous systems and training on very large quantities of data. (c) Out-of-core algorithms can handle vast quantities of data that cannot fit in a computer's main memory. An out-of-core learning algorithm chops the data into mini-batches and uses online learning techniques to learn from these mini-batches.

**04. (a) If your model performs great on the training data but generalizes poorly to new instances, what is happening? Can you name three possible solutions? (b) What is a test set, and why would you want to use it? (c) What is the purpose of a validation set?**

**Ans:** (a) If a model performs great on the training data but generalizes poorly to new instances, the model is likely overfitting the training data (or we got extremely lucky on the training data). Possible solutions to overfitting are getting more data, simplifying the model (selecting a simpler algorithm, reducing the number of parameters or features used, or regularizing the model), or reducing the noise in the training data. (b) A test set is used to estimate the generalization error that a model will make on new instances before the model is launched in production. (c) A validation set is used to compare models. It makes it possible to select the best model and tune the hyperparameters.

**05. Describe Newton's method for deriving the equation for finding the minimum (or maximum) of a given equation.**

**Ans**: Say we have an equation $f(x) = 0$

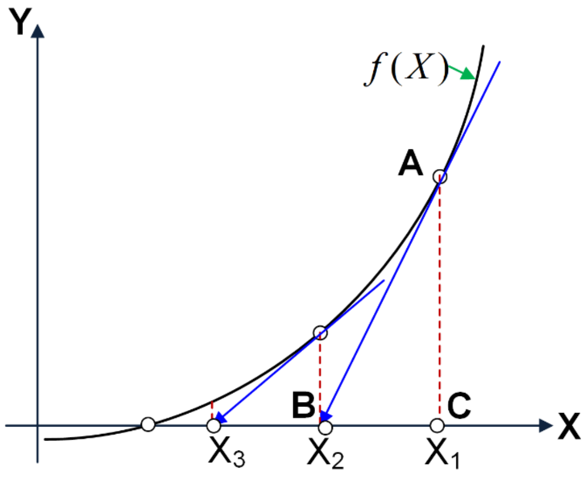


**Figure**: How Newton's method works in finding the solution.

For the solution of the equation (i.e., to find for what value of *x*, *f*(*x*) = 0), assume our initial point is *x₁,* which intersects the *x*-axis at C and *f*(*x*) at A (see Figure above). Also, consider that the tangent at A intersects the *x*-axis at B, where the value of *x* is $x_2$. From ΔABC and the definition of the slope of an equation, we can write:

$$f'(x_1) = \frac{f(x_1) - 0}{x_1 - x_2} \qquad (i)$$

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} \qquad (ii)$$

In general, we can write,

$$x_{t+1} = x_t - \frac{f(x_t)}{f'(x_t)} \qquad (iii)$$

Equation (*iii*) can be iteratively used to get the solution of the equation.

Now, for a minimization problem, if a minimum exists, then we need to find the value of *x* for which $f'(x) = 0$.

We can think of going for the solution of the equation, $f'(x) = 0$ using (*iii*).

Therefore, we can similarly write,

$$x_{t+1} = x_t - \frac{f'(x_t)}{f''(x_t)} \qquad (iv)$$

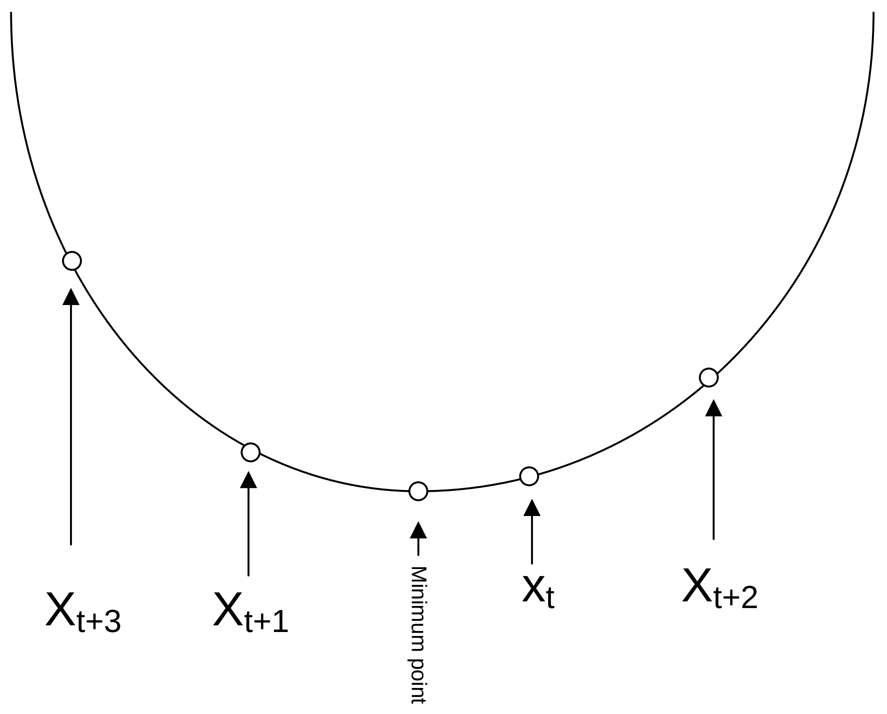
Now, we can use equation (*iv*) to find the minimum (or, maximum) of equation *f*(*x*) = 0.

**06. (a) What is the gradient descent equation? What is the $\alpha$ in that equation used for? (b) How might the overshooting problem occur with a gradient descent approach? (c) How does the gradient ascent equation differ from the descent equation?**

**Ans: (a)** [do it by yourself]

**Ans: (b) Hints:**
If we set the value of $\alpha$ (alpha, the learning rate), to a higher value then overshooting might occur. How?

**(A) <----------- (B)**



Say we are after the minimum point.
Assume we are at X(t) now.

We set $\alpha$ a very high value and compute x(t+1) as:

$$x_{t+1} = x_t - \alpha \ \nabla f(x_t)$$

Note that the $\alpha$ is constant – has a fixed value or it does not change in the iteration but remember slope $\nabla f(x_i)$ dependent on the location of $x_i$ and it changes.

Now, x(t+1) surpasses the minimum point when moving from **(B) to (A) direction** for the minimum point for a very higher deduction (amount: $\alpha \ \nabla f(x_t)$) from x(t).

From the figure, we see the distance of x(t+1) from the minimum point is higher than the distance of x(t) from the minimum point. So, it is obvious that:

$$|\nabla f(x_{t+1})| > |\nabla f(x_t)|$$

[in this case, $\nabla f(x_t)$ is +ve and $\nabla f(x_{t+1})$ is -ve]

$\alpha |\nabla f(x_{t+1})| > \alpha |\nabla f(x_t)|$ will be true.

Based on this information and the figure we can say that the next point x(t+2) would be behind x(t) as we will apply:

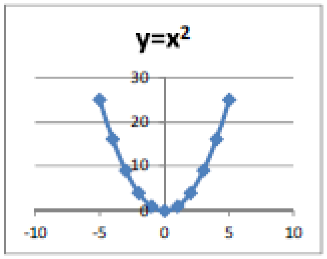
$$x_{t+2} = x_{t+1} - \alpha \ \nabla f(x_{t+1})$$

Again $\alpha\,|\nabla f(x_{t+2})| > \alpha\,|\nabla f(x_{t+1})| > \alpha\,|\nabla f(x_t)|$ will be true, so the next point x(t+3) will be behind x(t+1), etc.

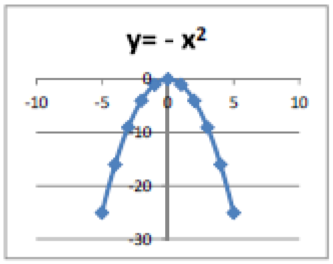
So, we see instead of converging, it is diverging in each iteration => overshooting.

**Ans: (c) Hints**:
Gradient descent (climbing down) is given by: $x_{t+1} = x_t - \alpha\ \nabla f(x_t)$
Gradient ascent (climbing up) is given by: $x_{t+1} = x_t + \alpha\ \nabla f(x_t)$

**(A) <----------- (B)** **(A) <----------- (B): direction**



(a)



(b)

**Figure**: Curve has (a) minimum point and (b) maximum point.

For curve (a) when the iteration goes from **(B) to (A)** to get the minimum point the slope (i.e., $\nabla f$ or, $f'(x)$) remains +ve. And we set the $\alpha$ (alpha; the learning rate) as a positive fixed number. So to reduce the $x_t$ in each iteration to move from **(B) to (A)** the term '$\alpha\ \nabla f(x_t)$' must be deducted.

Now, for curve (b), we cannot find a minimum (the minimum point is not there), rather, we have a maximum point. Using the equation iteratively, if we are moving from **(B) to (A)**, the slope in the case will be –ve, and $\alpha$ is positive, so the term $\alpha\ \nabla f(x_t)$ is now negative. So, we need to have a +ve sign in front of the term to keep it negative so, that the $x_t$ is deducted in each iteration to move it from **(B) to (A) direction**.

## 07. (a) How does the performance of the Gradient Descent vary depending on the learning rate (α)? (b) What are the names of the three variants of the Gradient Descent algorithm?

**Ans**: (a) The performance of GD depends on learning rate:

- If the learning rate is too low: the algorithm will eventually reach the solution, but it will take a long time.
- If the learning fits well, then in just a few iterations, GD converges to the solution.
- If the learning rate is too high: the algorithm diverges, jumping all over the place and getting further and further away from the solution at every step.

To find a good learning rate, you can use a grid search. However, you may want to limit the number of iterations so that the grid search can eliminate models that take too long to converge. To do that, we can apply a coarse-grain search first and then use the fine-grain search on the potential subpart of the grid found in the coarse-grain search.

(b) The three variants are:

- Batch Gradient Descent
- Stochastic Gradient Descent
- Mini-batch Gradient Descent

## 08. Show that the parameter $\beta$ is shrinking, given the gradient descent approach for regularization is:

$$\beta_j(t+1) = \beta_j(t) + \frac{2\alpha}{N}\left[\sum_{i=1}^{N} \left(y(i) \ - x^T(i)\,\beta\right).\, x(i)_j - \lambda\beta(t)_j\right]$$

**Ans**: It is given that,

$$\beta_j(t+1) = \beta_j(t) + \frac{2\alpha}{N}\left[\sum_{i=1}^{N} \left(y(i) \ - x^T(i)\,\beta\right).\, x(i)_j - \lambda\beta(t)_j\right]$$

And by rearranging, we can rewrite:

$$\beta_j(t+1) = \beta_j(t) - \frac{2\alpha}{N}.\lambda\beta(t)_j + \frac{2\alpha}{N}\left[\sum_{i=1}^{N} \left(y(i) \ - x^T(i)\,\beta\right).\, x(i)_j\right]$$

$$=> \beta_j(t+1) = \beta_j(t)(1 - \frac{2\alpha\lambda}{N}) + \frac{2\alpha}{N}\left[\sum_{i=1}^{N} \left(y(i) \ - x^T(i)\,\beta\right).\, x(i)_j\right]$$

We see the $\beta_j(t)$ is actually shrinking due to the term $(1-\frac{2\alpha\lambda}{N})$ , which is < 1 because $\alpha, \lambda$ and $N$ are positive quantities.

## 09. Write down the steps for k-fold cross-validation for model selection.

Ans:

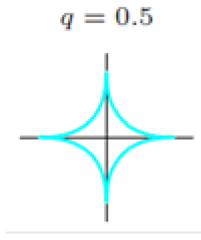
```
1. Randomly split data (D) into k disjoint subsets as: $D_1, D_2, ..., D_k$ .
2. For $\forall i$ , model $M_i$ is evaluated as:
       For $j$ =1 to k:
           Train the $M_i$ using data: ( $D - D_j$ ) and get the predictor $P_{ij}$
           Test $P_{ij}$ using $D_j$ and get the error $E_{ij}$.
         EndFor $j$

       $E_i$ = average of $E_{ij}$ for $\forall j$ , which is the generalized error of $M_i$.

3. Pick the best model $M_i$ having the lowest generalized error $E_i$.
4. Retrain $M_{i = \text{best}}$ using full dataset D.
```

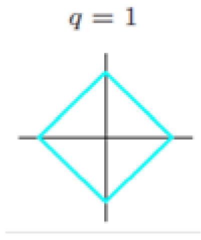
**10. RSS (β) in terms of a more general regularizer is given as:**

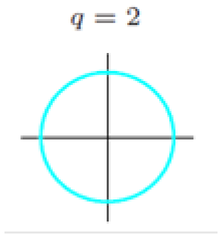
$$RSS(\beta) = \sum_{i=1}^{N} \{(\hat{y}(x_i, \beta) - y_i)^2\} + \lambda \sum_{j=1}^{p} |\beta_j|^q$$

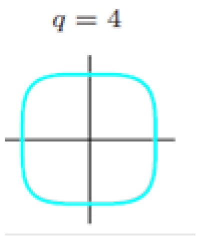
**Draw the contours of the regularization term when q = (a) 0.5, (b) 1, (c) 2 and (d) 4.**

Ans:

(a) 

(b) 

(c) 

(d) 

**11. Explain the "no free lunch theorem."**

Ans:

Much of machine learning is concerned with devising different models and different algorithms to fit them. We can use methods such as cross-validation to empirically choose the best method for our problem. However, there is no universally best model, sometimes called the no-free lunch theorem (Wolpert 1997). This is because a set of assumptions that work well in one domain may work poorly in another.

**12. Define the following terms: (a) accuracy, (b) sensitivity, (c) specificity, (d) precision, (e) balanced accuracy, and (f) MCC.**

**Ans**: Here, in all cases, *TP* stands for True Positive, *TN* stands for True Negative, *FP* stands for False Positive, and *FN* stands for False Negative.

**(a)** ***Accuracy*** is then defined as the sum of the number of true positives and true negatives divided by the total number of examples (where # indicates 'number of,' and TP stands for True Positive, etc.):

$$\boldsymbol{Accuracy} = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ prediction\ made} = \frac{TP + TN}{TP + FP + TN + FN}$$

**(b)** ***Sensitivity*** (also known as the ***true positive rate***, ***<u>recall</u>***, ***probability of detection, hit rate***) is the ratio of positive data points that are correctly predicted as positive with respect to all positive data points.

$$Sensitivity = \frac{TP}{TP + FN}$$

**(c)** ***Specificity*** (also known as the ***true negative rate***, ***selectivity***) is the ratio of the negative data points that are correctly predicted as negative, with respect to all negative data points.

$$Specificity = \frac{TN}{TN + FP}$$

**(d)** ***Precision*** (also known as the ***positive predictive value***) is the number of correct positive results divided by the number of positive results predicted by the classifier. That is, precision is defined as the accuracy of the judgment.

$$Precision = \frac{TP}{TP + FP}$$

**(e)** ***Balanced Accuracy***: In the case of imbalanced data, we can compute the ***balanced accuracy*** as the arithmetic mean of sensitivity and specificity.

$$Balanced\ Accuracy = \frac{(Sensitivity + Specificity)}{2} = \frac{1}{2}\left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP}\right)$$

**(f) MCC stands for** Matthew's Correlation Coefficient. In the case of imbalanced data, we can compute MCC. MCC is computed as:

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

**13. (a) What is unsupervised learning? (b) Given the square matrix A= $\begin{pmatrix} 3 & 2 \\ 2 & 6 \end{pmatrix}$, show your detailed works to compute the corresponding Eigen-values and their corresponding Eigen-vectors.**

**Ans: (a) [Chapter #4, Page 1].**

**Ans: (b)**

We know Av = λv, where v is the non-zero column vector, and λ is the corresponding Eigen-value.

Therefore, using the given Matrix of A, from Av = λv, we can write:

⇨ (A-λI) ***v*** = 0 … … … … … … … … … … … … … … … … … … … (G)

[I = identity matrix, we need to introduce I since λ is scalar]

⇨ $(A-\lambda I) = 0$ [∵ ***v*** is a non-zero matrix]

⇨ $\begin{pmatrix} 3 & 2 \\ 2 & 6 \end{pmatrix} - \begin{pmatrix} \lambda & 0 \\ 0 & \lambda \end{pmatrix} = 0$

⇨ $\begin{pmatrix} 3-\lambda & 2 \\ 2 & 6-\lambda \end{pmatrix} = 0$

⇨ $(3-\lambda)(6-\lambda)-4=0$

⇨ $\lambda^2 - 9\lambda + 14 = 0$

⇨ $\lambda = 7, 2$; being larger $\lambda = 7$ is the 1st principal eigenvalue.

Now, the question is, what would be the corresponding ***v***s. of $\lambda = 7, 2$.

Using $\lambda = 7$ and Equation (*i*):

$$\begin{pmatrix} 3 & 2 \\ 2 & 6 \end{pmatrix}\begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = 7\begin{pmatrix} v_1 \\ v_2 \end{pmatrix}$$

=>

⇨ we get: $3v_1 + 2v_2 = 7v_1$ and $2v_1 + 6v_2 = 7v_2$

⇨ From both equations, we get: $2\boldsymbol{v}_1 = \boldsymbol{v}_2$ => $\frac{v_1}{v_2} = \frac{1}{2}$ => $\boldsymbol{v} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$.

Further, to make it a unit vector, we can divide its components by $\sqrt{5}$. Thus, the principal (unit) eigenvector can be $\begin{pmatrix} \frac{1}{\sqrt{5}} \\ \frac{2}{\sqrt{5}} \end{pmatrix}$.

Similarly, using $\lambda = 2$:

We get, (unit) eigenvector $\begin{pmatrix} \frac{2}{\sqrt{5}} \\ -\frac{1}{\sqrt{5}} \end{pmatrix}$

**14. Write down the PCA algorithm.**
**Ans:**

**Algorithm:** PCA

**Inputs**: Parameter $q$ (desired lower dimension) where, $q < p$,
dataset D $\{(x_1), \ldots, (x_N)\}$.

```
1. For j=1: p        // Compute the means or centroids of all the p features
```

```
        mean, $\mu_j = \frac{1}{N}\sum_{i=1}^{N} x_{ij}$
    End
2. For all, update $x = (x - \mu)$    // this is to have 0 means.
3. Scale, $x = \frac{x}{\sigma}$    // where, $\sigma$ is the standard deviation. This step is
                                     needed when the features are in different scales.
4. Compute, Sigma (S) = $\frac{1}{N}\sum_{i=1}^{N} x_i x_i^T$
5. [U,M,V] =svd (Sigma);    // Octave/MATLAB function
6. $U_q$ = U(:, 1:q);
7. $\tilde{x} = U_q^T * x$;  // Projected approximate values ( $\tilde{x}$ ) are computed.
```

## 15. What is Lagrange-multiplier or Lagrange-optimization? Describe using an example.

**Ans**: *Lagrange Multipliers*, also known as *undetermined multipliers*, are used to find the stationary points of a function of several variables subject to one or more constraints.

Consider the problem of finding the maximum of a function $f(x_1, x_2)$ subject to a constraint relating $x_1$ and $x_2$, which we write in the form.

$$g(x_1, x_2) = 0 \tag{A.1}$$

To solve, a more elegant and often simpler approach is based on the introduction of a parameter λ called a *Lagrange multiplier*. Then, the problem can be solved by optimizing the *Lagrangian function*

$$L(\mathrm{x}, \lambda) \equiv f(\mathrm{x}) + \lambda\, g(\mathrm{x}) \tag{A.2}$$

We then differentiate A.2 with respect to the variables (i.e., x, λ) and equate the equation(s) to zero, and then we get others equations, using which we can get suitable solutions.

As an example, suppose we wish to find the stationary points of the function $f(x_1, x_2) = 1 - x_1^2 - x_2^2$ subject to the constraint $g(x_1, x_2)$ => $x_1 + x_2 - 1 = 0$. The corresponding *Lagrangian function* is given by:

$$L(\mathrm{x}, \lambda) = 1 - x_1^2 - x_2^2 + \lambda\,(x_1 + x_2 - 1) \tag{A.3}$$

The conditions for this *Lagrangian* to be stationary with respect to $x_1$ , $x_2$ and λ give the following coupled equations:

$$-2x_1 + \lambda = 0 \tag{A.4}$$

$$-2x_2 + \lambda = 0 \quad (A.5)$$
$$x_1 + x_2 - 1 = 0 \quad (A.6)$$

The solution of these equations then gives the stationary point as $(x_1^*, x_2^*) = \left(\frac{1}{2}, \frac{1}{2}\right)$ and the corresponding value for the Lagrange multiplier is $\lambda = 1$.

**16. (a) What are manifold and manifold learning? (b) describe manifold learning in terms of machine learning (c) How does Locally Linear Embedding (LLE) work?**

**Ans:** (a) A d-dimensional manifold is a d-dimensional shape that can be bent and twisted in a higher-dimensional space. More generally, a d-dimensional manifold is part of an n-dimensional space (where d < n) that resembles a d-dimensional hyperplane locally. Many dimensionality reduction algorithms work by modeling the manifold on which the training instances lie; this is called manifold learning. It relies on the manifold assumption, also called the manifold hypothesis, which holds that most real-world high-dimensional datasets lie close to a much lower-dimensional manifold.

(b) In terms of machine learning, although the data points may consist of thousands of features, they may be described as a function of only a few underlying parameters. That is, the data points are actually samples from a low-dimensional manifold that is embedded in a high-dimensional space. Manifold learning algorithms attempt to uncover these parameters in order to find a low-dimensional representation of the data. Some approaches to solving this problem include Isomap, Locally Linear Embedding, Laplacian Eigenmaps, Semidefinite Embedding, etc. These algorithms work towards extracting the low-dimensional manifold that can be used to describe the high-dimensional data.

(c) Locally Linear Embedding (LLE) is a Manifold Learning technique that does not rely on projections. LLE works by first measuring how each training instance linearly relates to its closest neighbors (c.n.) and then looking for a low-dimensional representation of the training set where these local relationships are best preserved. This approach makes it particularly good at unrolling twisted manifolds, especially when there is not too much noise.

**17. (a) How does face detection differ from face recognition? (b) what is the vision? CascadeObjectDetector refer to in MATLAB?**

**Ans:** (a) Face detection refers to the process of identifying human faces in digital images, whereas face recognition is a process of identifying an individual by comparing a given image with the previously recorded image of that person. However, face detection can be an important preprocessing step of the face-recognition approach from given random images.

(b) The cascade object detector is a series of models in the AdaBoost algorithm, which uses the Viola-Jones algorithm to detect people's faces, noses, eyes, mouths, or upper body.

**18. (a) What was the motivation behind the boosting algorithm? (b) Write down the steps of the AdaBoost.M1 Algorithm (chapter 6, see the slides)**

Ans:

(a) The motivation for boosting was a procedure that combines the outputs of many "weak" classifiers to produce a powerful "committee."

(b) The AdaBoost.M1 algorithm is given as follows:

**Algorithm:** AdaBoost.M1.

```
1. Initialize the observation weights $w_i = 1/N$, i = 1, 2, …, $N$.
2. For $m = 1$ to $M$
        (a) Fit a classifier $G_m(x)$ to the training data using weights $w_i$.
        (b) Compute
                $err_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}$
        (c) Compute $\alpha_m = \log((1 - err_m)/err_m)$.
        (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1,2,\dots,N$.
3. Output $G(x) = \text{sign}\,[\sum_{m=1}^{M} \alpha_m G_m(x)]$.
```

**19. (a) What are the major stages of the Viola-Jones Algorithm for face detection? (b) Define integral image. (c) What is/are the formula/s to compute the integral image? (d) Using at least one go-through step, compute the integral image of the following image-matrix:**

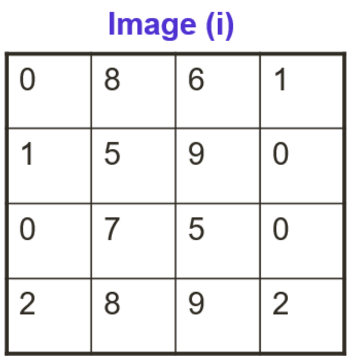


Image (i)

| | | | |
|---|---|---|---|
| 0 | 8 | 6 | 1 |
| 1 | 5 | 9 | 0 |
| 0 | 7 | 5 | 0 |
| 2 | 8 | 9 | 2 |

**Ans:**

**(a)** The major steps are:

- <u>Features</u>: Feature generation/extraction using Haar-like feature set.

- Integral Image: A novel image presentation scheme helps compute features faster.
- Feature Classifier: Boost the learning using AdaBoost.
- Cascading: Reject negative class fast, but thoroughly compute potential positive class - thus enhance the robustness.

**(b)** We define the integral image (*ii*) at location *x*, *y* contains the sum of the pixels above and to the left of *x*, *y*, inclusive as indicated in equation (C) and figure #A:

$$ii(x,\ y) = \sum_{x' \leq x,\ y' \leq y} i(x',\ y') \qquad \text{(C)}$$

where, *ii* (*x*, *y*) is the integral image, and *i* (*x'*, *y'*) is the original image.

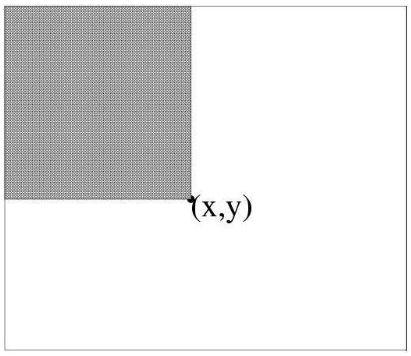


**Figure A**: The value of the integral image at point (*x*, *y*) is the sum of all the pixels above and to the left.

**(c)** Integral image, *ii* (*x*, *y*) can be computed using the following pair of recurrences:

$$s(x, y) = s(x, y - 1) + i(x, y) \qquad \text{(D)}$$

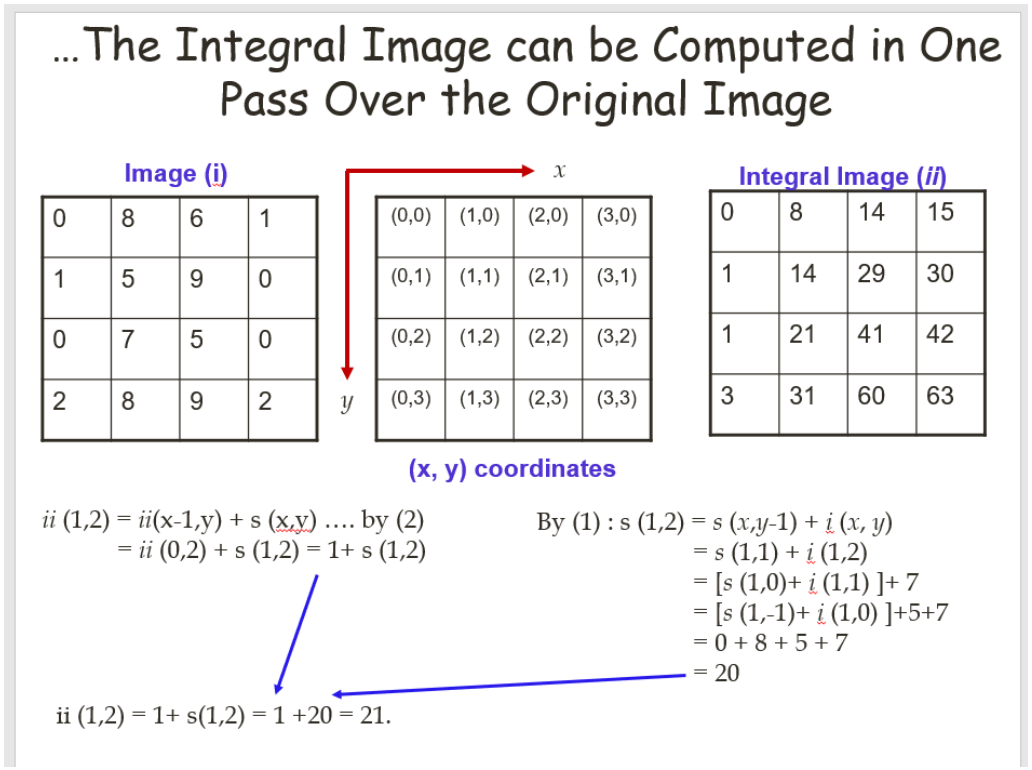
$$ii(x, y) = ii(x - 1, y) + s(x, y) \qquad \text{(E)}$$

where, $s(x, y)$ is the cumulative row sum,
$s(x,-1) = 0$, and
$ii(-1, y) = 0$

Or, the Integral image *ii* (*x*, *y*) can be computed using the following equation (F):

$$ii(x, y) = [\, i(x, y) + ii(x-1, y) + ii(x, y-1) \,] - ii(x-1, y-1) \qquad \text{(F)}$$

**(d)** [Hints: use slide #(19) or slide #(20) of chapter 06]



## …The Integral Image can be Computed in One Pass Over the Original Image

Image (i)

|   |   |   |   |
|---|---|---|---|
| 0 | 8 | 6 | 1 |
| 1 | 5 | 9 | 0 |
| 0 | 7 | 5 | 0 |
| 2 | 8 | 9 | 2 |

(x, y) coordinates

|   |   |   |   |
|---|---|---|---|
| (0,0) | (1,0) | (2,0) | (3,0) |
| (0,1) | (1,1) | (2,1) | (3,1) |
| (0,2) | (1,2) | (2,2) | (3,2) |
| (0,3) | (1,3) | (2,3) | (3,3) |

Integral Image (*ii*)

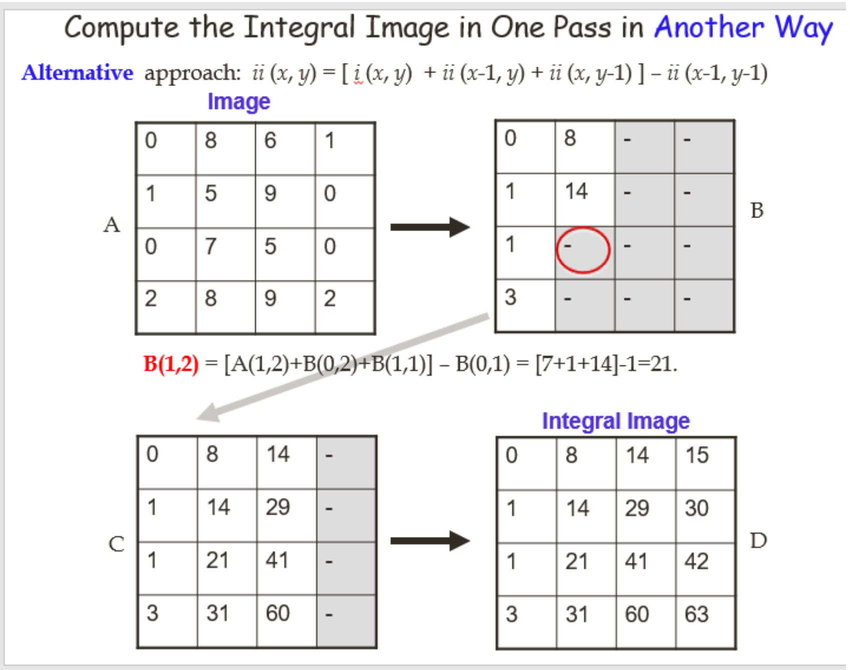
|   |   |   |   |
|---|---|---|---|
| 0 | 8 | 14 | 15 |
| 1 | 14 | 29 | 30 |
| 1 | 21 | 41 | 42 |
| 3 | 31 | 60 | 63 |

*ii* (1,2) = *ii*(x-1,y) + s (x,y) …. by (2)
= *ii* (0,2) + s (1,2) = 1+ s (1,2)

By (1) : s (1,2) = *s* (*x*,*y*-1) + *i* (*x*, *y*)
= *s* (1,1) + *i* (1,2)
= [*s* (1,0)+ *i* (1,1) ]+ 7
= [*s* (1,-1)+ *i* (1,0) ]+5+7
= 0 + 8 + 5 + 7
= 20

ii (1,2) = 1+ s(1,2) = 1 +20 = 21.

**Or,**



Compute the Integral Image in One Pass in Another Way

Alternative approach: $ii(x, y) = [\,i(x, y) + ii(x-1, y) + ii(x, y-1)\,] - ii(x-1, y-1)$

Image

A

| | | | |
|---|---|---|---|
| 0 | 8 | 6 | 1 |
| 1 | 5 | 9 | 0 |
| 0 | 7 | 5 | 0 |
| 2 | 8 | 9 | 2 |

B

| | | | |
|---|---|---|---|
| 0 | 8 | - | - |
| 1 | 14 | - | - |
| 1 | - | - | - |
| 3 | - | - | - |

B(1,2) = [A(1,2)+B(0,2)+B(1,1)] – B(0,1) = [7+1+14]-1=21.

C

| | | | |
|---|---|---|---|
| 0 | 8 | 14 | - |
| 1 | 14 | 29 | - |
| 1 | 21 | 41 | - |
| 3 | 31 | 60 | - |

Integral Image

D

| | | | |
|---|---|---|---|
| 0 | 8 | 14 | 15 |
| 1 | 14 | 29 | 30 |
| 1 | 21 | 41 | 42 |
| 3 | 31 | 60 | 63 |

**Alternative approach**: $ii(x, y) = [\,i(x, y) + ii(x-1, y) + ii(x, y-1)\,] - ii(x-1, y-1)$

Question: How to compute B(1,2) (applying alternative way)?

```
B(1,2) = [A (1,2)+ B (1-1,2) +B (1,2-1)] – B(1-1,2-1)
       = [A(1,2)+B(0,2)+B(1,1)] – B(0,1)
       = [7+ 1+14]-1
       = 22-1
       = 21
```

## 20. What are the main tasks that autoencoders are used for?

**Ans**: Here are some of the main tasks that autoencoders are used for:

- Feature extraction
- Unsupervised pretraining
- Dimensionality reduction
- Generative models
- Anomaly detection (an autoencoder is generally bad at reconstructing outliers).

## 21. Suppose you want to train a classifier, and you have plenty of unlabeled training data but only a few thousand labeled instances. How can autoencoders help? How would you proceed?

**Ans**: If you want to train a classifier and you have plenty of unlabeled training data but only a few thousand labeled instances, then you could first train a deep autoencoder on the full dataset (labeled + unlabeled), then reuse its lower half for the classifier (i.e., reuse the layers up to the codings layer, included) and train the classifier using the labeled data. If you have little labeled data, you probably want to freeze the reused layers when training the classifier.

**22. What are the two major parts of the Generative Adversarial Network (GAN). Draw a schematic diagram of GAN.**

**Ans:**
(a) GANs are composed of two neural networks:
(*i*) a **generator** that tries to generate data that looks similar to the training data, and
(*ii*) a **discriminator** that tries to tell real data from fake data.

(b)

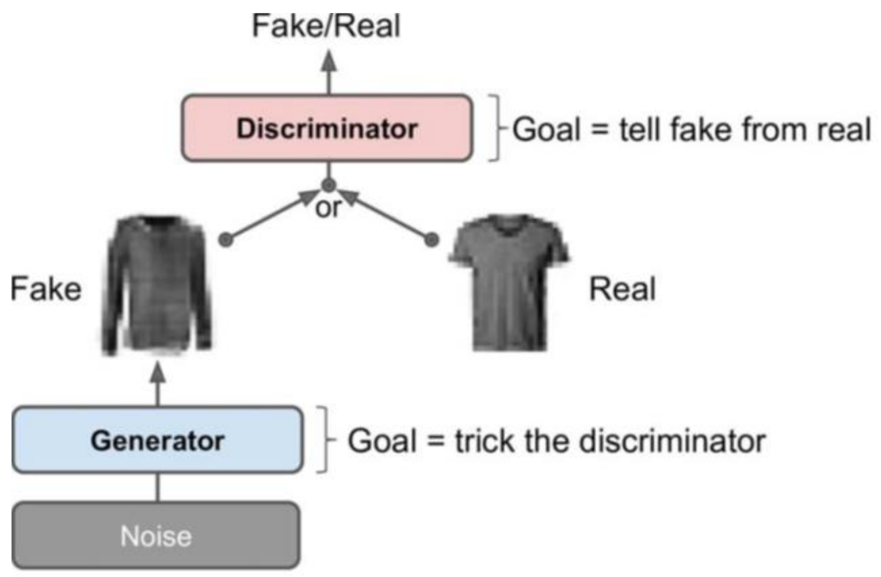


**Figure**: A schematic diagram of the generative adversarial network (GAN).

**23. What are the two classes of classical approaches for numerically representing words?**

**Ans:**

They are:
(*i*) approaches that use **external resources** for representing words, such as WordNet and
(*ii*) approaches that **do not use external resources**, such as one-hot encoding and Term Frequency-Inverse Document Frequency (TF-IDF).

==================== Chapter 08 [optional] ==========================

**The following questions are optional for the final exam. However, the questions could be in the final exam as bonus questions:**

**24. Can you think of a few applications for a sequence-to-sequence RNN? What about a sequence-to-vector RNN and a vector-to-sequence RNN?**

**Ans**: Here are a few RNN applications:

• For a sequence-to-sequence RNN: predicting the weather (or any other time series), machine translation (using an Encoder-Decoder architecture), video captioning, speech to text, music generation (or other sequence generation), identifying the chords of a song.

• For a sequence-to-vector RNN: classifying music samples by music genre, analyzing the sentiment of a book review, predicting what word an aphasic patient is thinking of based on readings from brain implants, predicting the probability that a user will want to watch a movie based on their watch history (this is one of many possible implementations of collaborative filtering for a recommender system).

• For a vector-to-sequence RNN: image captioning, creating a music playlist based on an embedding of the current artist, generating a melody based on a set of parameters, locating pedestrians in a picture (e.g., a video frame from a self-driving car's camera).

**25. How many dimensions must the inputs of an RNN layer have? What does each dimension represent? What about its outputs?**

**Ans**: An RNN layer must have three-dimensional inputs: the first dimension is the batch dimension (its size is the batch size), the second dimension represents the time (its size is the number of time steps), and the third dimension holds the inputs at each time step (its size is the number of input features per time step). For example, if you want to process a batch containing 5 time series of 10 time steps each, with 2 values per time step (e.g., the temperature and the wind speed), the shape will be [5, 10, 2]. The outputs are also three-dimensional, with the same first two dimensions, but the last dimension is equal to the number of neurons. For example, if an RNN layer with 32 neurons processes the batch we just discussed, the output will have a shape of [5, 10, 32].

**26. If you want to build a deep sequence-to-sequence RNN, which RNN layers should have return_sequences=True? What about a sequence-to-vector RNN?**

**Ans**: To build a deep sequence-to-sequence RNN using Keras, you must set return_sequences=True for all RNN layers. To build a sequence-to-vector RNN, you must set return_sequences=True for all RNN layers except for the top RNN layer, which must have return_sequences=False (or do not set this argument at all, since False is the default).

**27. Suppose you have a daily univariate time series, and you want to forecast the next seven days. Which RNN architecture should you use?**

**Ans**: If you have a daily univariate time series and you want to forecast the next seven days, the simplest RNN architecture you can use is a stack of RNN layers (all with return_sequences=True except for the top RNN layer), using seven neurons in the output RNN layer. You can then train this model using random windows from the time series (e.g., sequences of 30 consecutive days as the inputs and a vector containing the values of the next 7 days as the target). This is a sequence-to-vector RNN. Alternatively, you could set return_sequences=True for all RNN layers to create a sequence-to-sequence RNN. You can train this model using random windows from the time series, with sequences of the same length as the inputs as the targets. Each target sequence should have seven values per time step (e.g., for time step t, the target should be a vector containing the values at time steps t + 1 to t + 7).

**28. What are the main difficulties when training RNNs? How can you handle them?**
**Ans**: The two main difficulties when training RNNs are unstable gradients (exploding or vanishing) and a very limited short-term memory. These problems both get worse when dealing with long sequences. To alleviate the unstable gradients problem, you can use a lower learning rate, use a saturating activation function such as the hyperbolic tangent (which is the default), and possibly use gradient clipping, Layer Normalization, or dropout at each time step. To tackle the limited short-term memory problem, you can use LSTM or GRU layers (this also helps with the unstable gradients problem).

**---------------------------------------- x ----------------------------------------**