SECA4002 – DEEP LEARNING NEURAL NETWORKS

Dr. V. Vedanarayanan B.E., M.E., PhD
Course Co-ordinator
Assistant Professor, Department of ECE,
School of Electrical and Electronics
SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY

Course Outcomes

SECA4002 – DEEP LEARNING NEURAL NETWORKS

At The end of this Course, Student will be able to

CO1 Select suitable model parameters for different machine learning techniques

CO2 Evaluate the performance of existing deep learning models for various applications

CO3 Realign high dimensional data using reduction techniques

CO4 Analyze the performance of various optimization and generalization techniques in deep learning

CO5 Modify the existing architectures for domain specific applications

CO6 Develop a real time application using deep learning neural networks

Course Objectives

SECA4002 – DEEP LEARNING NEURAL NETWORKS

Course Objectives:

- To present the mathematical, statistical and computational challenges of building neural networks
- To study the concepts of deep learning
- To introduce dimensionality reduction techniques
- To enable the students to know deep learning techniques to support real-time applications
- To examine the case studies of deep learning techniques

SECA4002 – DEEP LEARNING NEURAL NETWORKS

Detailed Syllabus:

UNIT 2: DEEP NETWORKS

History of Deep Learning- A Probabilistic Theory of Deep Learning- Backpropagation and regularization, batch normalization- VC Dimension and Neural Nets-Deep Vs Shallow Networks Convolutional Networks- Generative Adversarial Networks (GAN), Semi-supervised Learning

SECA4002 – DEEP LEARNING NEURAL NETWORKS

Recommended Text Books/ Reference Books

- Cosma Rohilla Shalizi, Advanced Data Analysis from an Elementary Point of View, 2015.
- Deng & Yu, Deep Learning: Methods and Applications, Now Publishers, 2013.
- Ian Goodfellow, Yoshua Bengio, Aaron Courville, Deep Learning, MIT Press, 2016.
- Michael Nielsen, Neural Networks and Deep Learning, Determination Press, 2015.

SECA4002 – DEEP LEARNING NEURAL NETWORKS

Detailed Syllabus:

UNIT 2: DEEP NETWORKS

History of Deep Learning- A Probabilistic Theory of Deep Learning- Backpropagation and regularization, batch normalization- VC Dimension and Neural Nets-Deep Vs Shallow Networks Convolutional Networks- Generative Adversarial Networks (GAN), Semi-supervised Learning

Deep Learning - History

- ❑ The chain rule that underlies the back-propagation algorithm was invented in the seventeenth century (Leibniz, 1676; L'Hôpital, 1696)
- ❑ Beginning in the 1940s, the function approximation techniques were used to motivate machine learning models such as the perceptron
- ❑ The earliest models were based on linear models. Critics including Marvin Minsky pointed out several of the flaws of the linear model family, such as its inability to learn the XOR function, which led to a backlash against the entire neural network approach
- ❑ Efficient applications of the chain rule based on dynamic programming began to appear in the 1960s and 1970s
- ❑ Werbos (1981) proposed applying chain rule techniques for training artificial neural networks. The idea was finally developed in practice after being independently rediscovered in different ways (LeCun, 1985; Parker, 1985; Rumelhart et al., 1986a)
- ❑ Following the success of back-propagation, neural network research gained popularity and reached a peak in the early 1990s. Afterwards, other machine learning techniques became more popular until the modern deep learning renaissance that began in 2006
- ❑ The core ideas behind modern feedforward networks have not changed substantially since the 1980s. The same back-propagation algorithm and the same approaches to gradient descent are still in use.

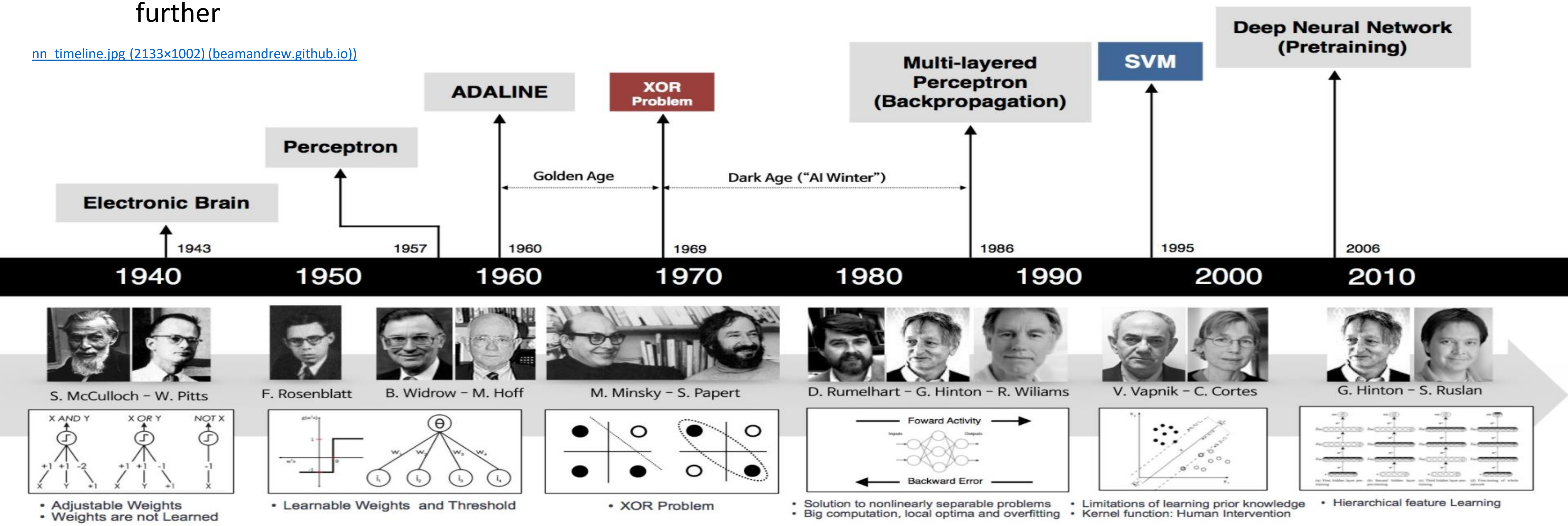
Deep Learning - History

- Most of the improvement in neural network performance from 1986 to 2015 can be attributed to two factors. First, larger datasets have reduced the degree to which statistical generalization is a challenge for neural networks. Second, neural networks have become much larger, because of more powerful computers and better software infrastructure.
- A small number of algorithmic changes have also improved the performance of neural networks noticeably. One of these algorithmic changes was the replacement of mean squared error with the cross-entropy family of loss functions. Mean squared error was popular in the 1980s and 1990s but was gradually replaced by cross-entropy losses and the principle of maximum likelihood as ideas spread between the statistics community and the machine learning community.
- The other major algorithmic change that has greatly improved the performance of feedforward networks was the replacement of sigmoid hidden units with piecewise linear hidden units, such as rectified linear units. Rectification using the max{0, z} function was introduced in early neural network models and dates back at least as far as the Cognitron and Neo-Cognitron (Fukushima, 1975, 1980).
- For small datasets, Jarrett et al. (2009) observed that using rectifying nonlinearities is even more important than learning the weights of the hidden layers. Random weights are sufficient to propagate useful information through a rectified linear network, enabling the classifier layer at the top to learn how to map different feature vectors to class identities. When more data is available, learning begins to extract enough useful knowledge to exceed the performance of randomly chosen parameters. Glorot et al. (2011a) showed that learning is far easier in deep rectified linear networks than in deep networks that have curvature or two-sided saturation in their activation functions.

Deep Learning - History

- ❖ When the modern resurgence of deep learning began in 2006, feedforward networks continued to have a bad reputation. From about 2006 to 2012, it was widely believed that feedforward networks would not perform well unless they were assisted by other models, such as probabilistic models. Today, it is now known that with the right resources and engineering practices, feedforward networks perform very well. Today, gradient-based learning in feedforward networks is used as a tool to develop probabilistic models.
- ❖ Feedforward networks continue to have unfulfilled potential. In the future, we expect they will be applied to many more tasks, and that advances in optimization algorithms and model design will improve their performance even further

nn_timeline.jpg (2133×1002) (beamandrew.github.io))



Electronic Brain
Perceptron
ADALINE
XOR Problem
Multi-layered Perceptron (Backpropagation)
SVM
Deep Neural Network (Pretraining)
Golden Age
Dark Age ("AI Winter")
1943
1957
1960
1969
1986
1995
2006
1940
1950
1960
1970
1980
1990
2000
2010
S. McCulloch – W. Pitts
F. Rosenblatt
B. Widrow – M. Hoff
M. Minsky – S. Papert
D. Rumelhart – G. Hinton – R. Wiliams
V. Vapnik – C. Cortes
G. Hinton – S. Ruslan
X AND Y
X OR Y
NOT X
Foward Activity
Backward Error
• Adjustable Weights
• Weights are not Learned
• Learnable Weights and Threshold
• XOR Problem
• Solution to nonlinearly separable problems
• Big computation, local optima and overfitting
• Limitations of learning prior knowledge
• Kernel function: Human Intervention
• Hierarchical feature Learning

SECA4002 – DEEP LEARNING NEURAL NETWORKS

Detailed Syllabus:

UNIT 2: DEEP NETWORKS

History of Deep Learning- A Probabilistic Theory of Deep Learning- Backpropagation and regularization, batch normalization- VC Dimension and Neural Nets-Deep Vs Shallow Networks Convolutional Networks- Generative Adversarial Networks (GAN), Semi-supervised Learning

Deep Learning – Probabilistic Theory

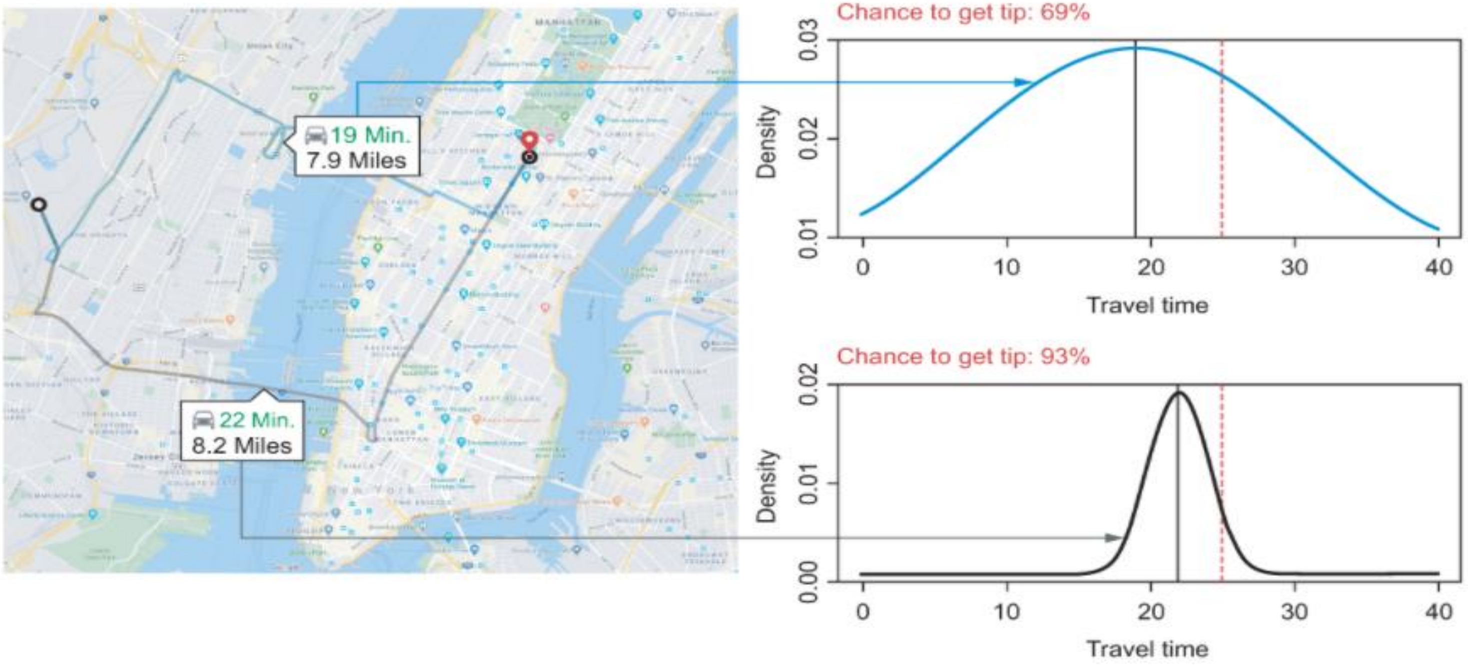
- ❑ Probability is the science of quantifying uncertain things.
- ❑ Most of machine learning and deep learning systems utilize a lot of data to learn about patterns in the data.

Whenever data is utilized in a system rather than sole logic, uncertainty grows up and whenever uncertainty grows up, probability becomes relevant.

- ❑ By introducing probability to a deep learning system, we introduce common sense to the system
- ❑ In deep learning, several models like Bayesian models, probabilistic graphical models, Hidden Markov models are used. They depend entirely on probability concepts.

❖ <u>Real world data is chaotic. Since deep learning systems utilize real world data, they require a tool to handle the chaoticness.</u>

You'll get $500 tip if I arrive at MoMA within 25 minutes!
Let's use my probabilistic travel time gadget!
TAXI



https://www.manning.com/books/probabilistic-deep-learning

Deep Learning – Probabilistic Theory

Figure 1.4 Chantal (left) has a large distance between the eyes and a rather small mouth. Sara (right) has a small distance between the eyes and a rather large mouth.

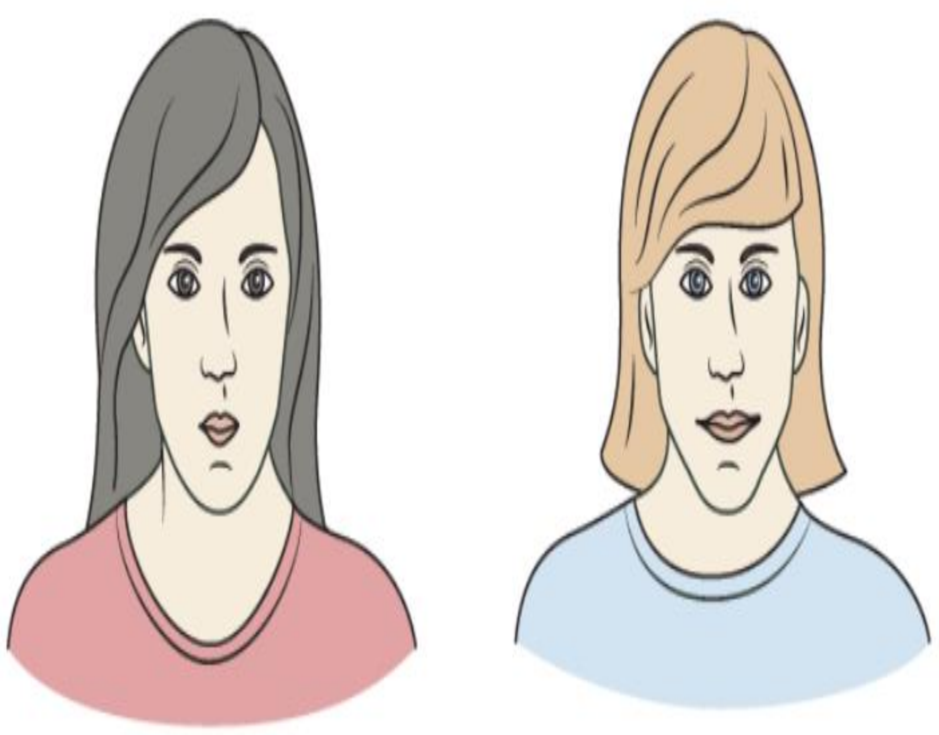
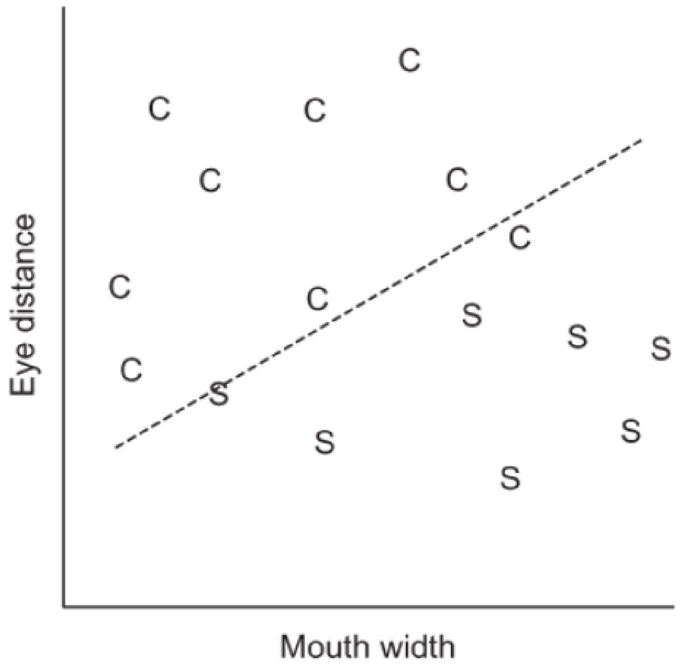


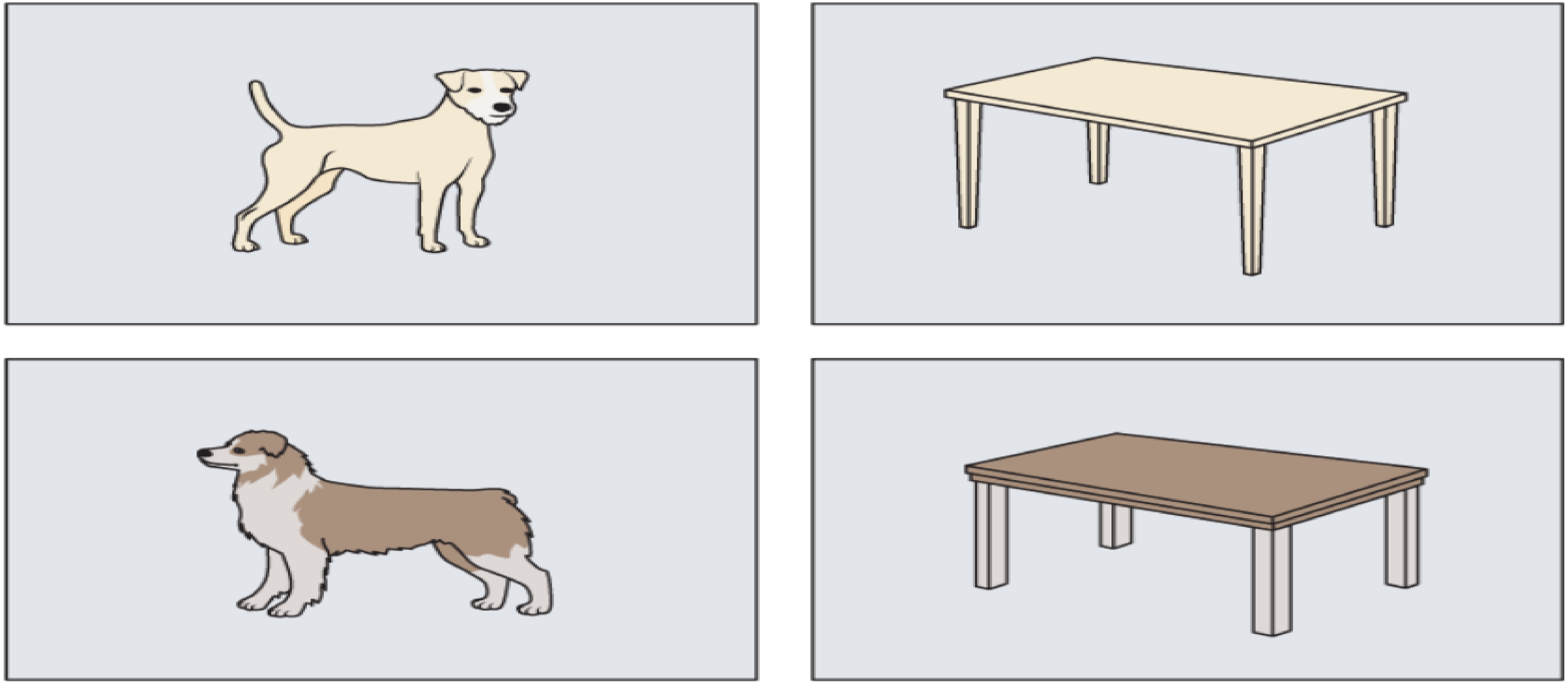
Figure 1.5 A 2D space spanned by the features mouth width and eye distance. Each point represents an image described by these two features (S for Sara and C for Chantal). The dashed line is a decision boundary separating the two classes.



https://www.manning.com/books/probabilistic-deep-learning

Deep Learning – Probabilistic Theory

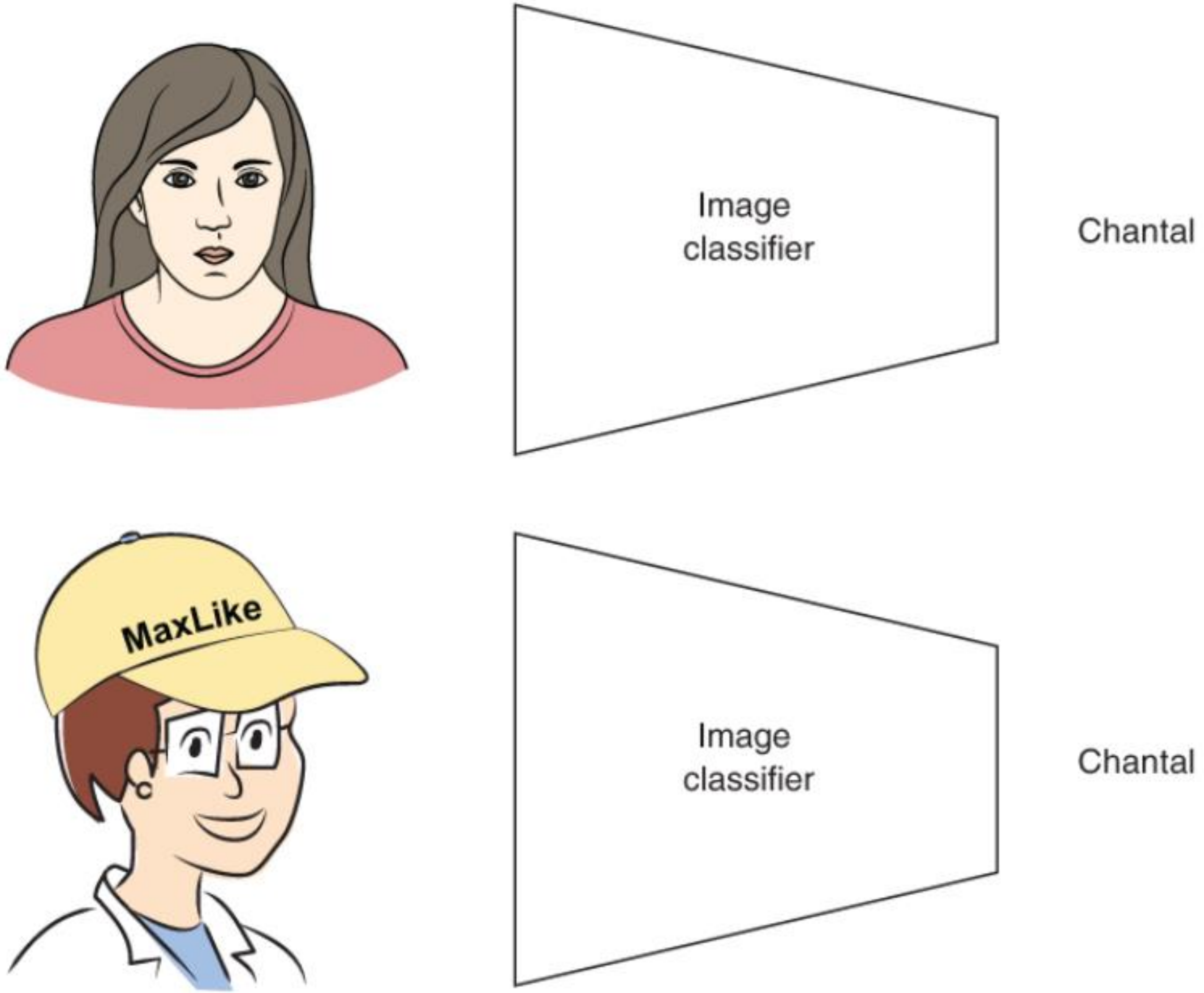
Figure 1.6 The left column shows two images of the class dog. The right column shows two images of the class table. When comparing the pictures on the pixel level, the two images in the same column are less similar than the two images in the same row, even if one image in a row shows a dog and the other image displays a table.



https://www.manning.com/books/probabilistic-deep-learning

Deep Learning – Probabilistic Theory

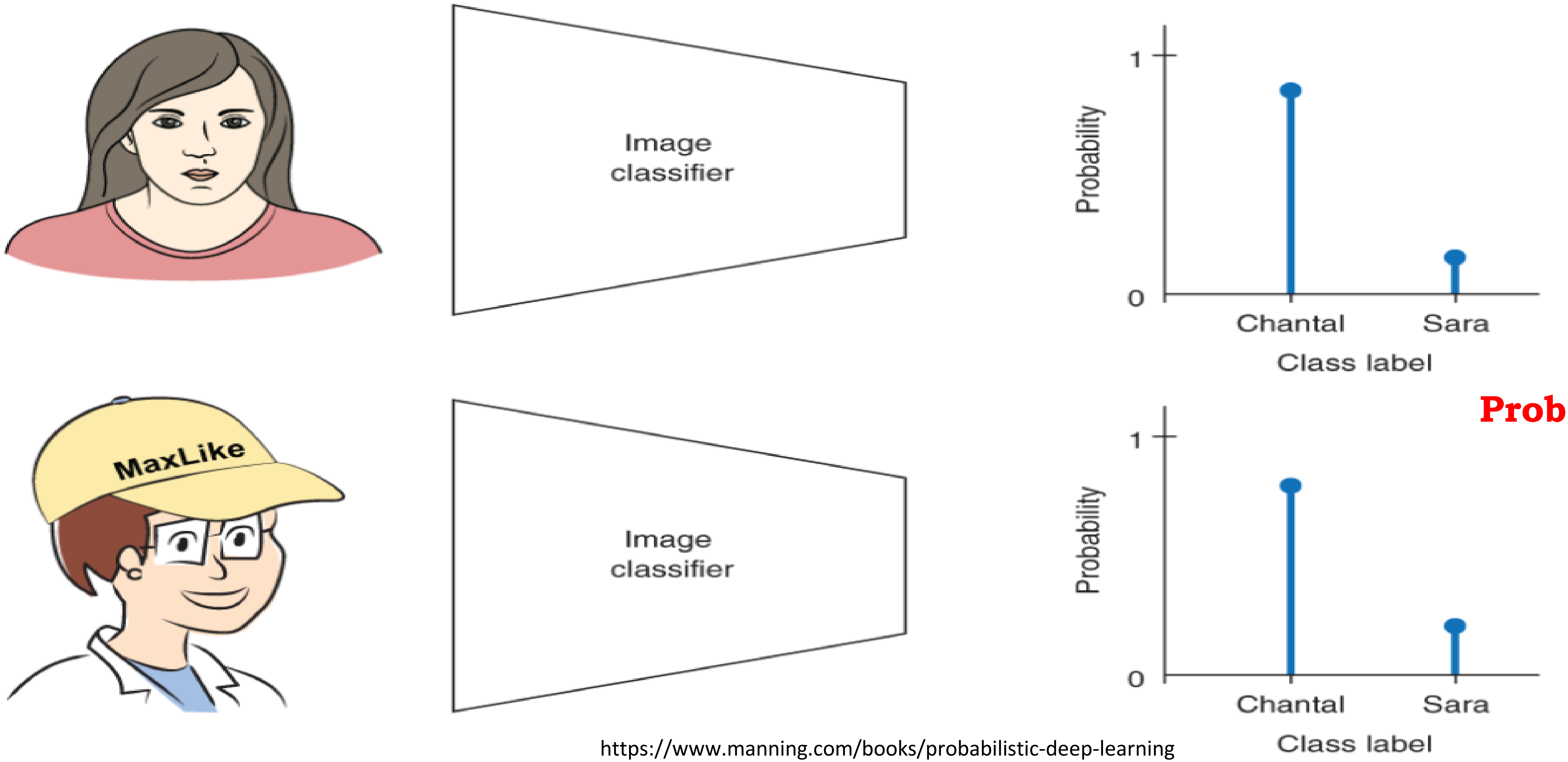
Figure 1.7 A non-probabilistic image classifier for face recognition takes as input an image and yields as outcome a class label. Here the predicted class label is Chantal, but only the upper image really shows Chantal. The lower image shows a woman who is neither Chantal nor Sara.



Non- Probabilistic Classifier

Deep Learning – Probabilistic Theory

Figure 1.8 A probabilistic image classifier for face recognition takes as input an image and yields as outcome a probability for each class label. In the upper panel, the image shows Chantal, and the classifier predicts a probability of 0.85 for the class Chantal and a probability of 0.15 for the class Sara. In the lower panel, the image shows neither Chantal nor Sara, and the classifier predicts a probability of 0.8 for the class Chantal and a probability of 0.2 for the class Sara.

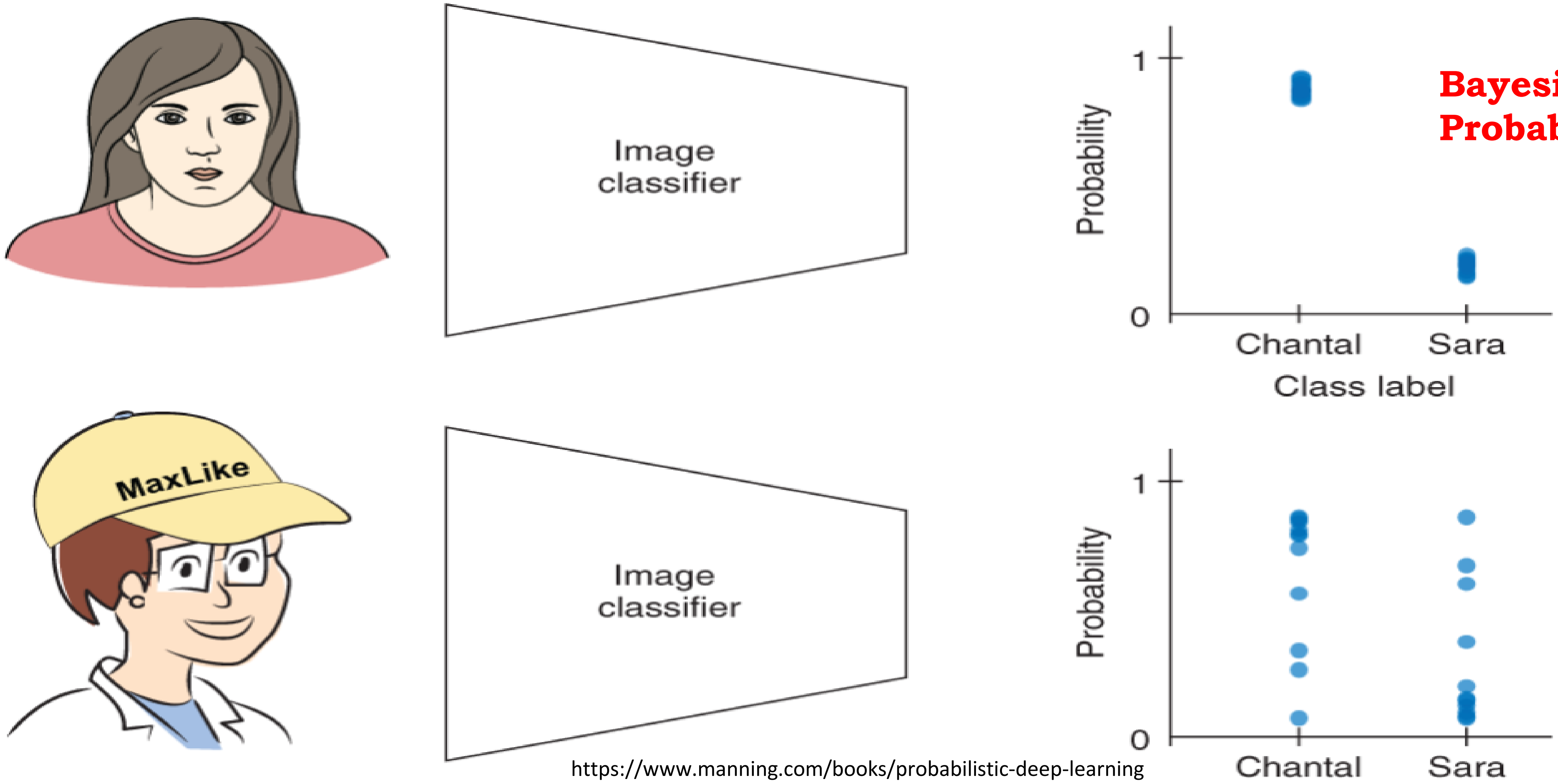


Probabilistic Classifier

https://www.manning.com/books/probabilistic-deep-learning

Deep Learning – Probabilistic Theory

Figure 1.9 A Bayesian probabilistic image classifier for face recognition takes as input an image and yields as outcome a distribution of probability sets for the two class labels. In the upper panel, the image is showing Chantal, and the predicted sets of probabilities all predict a large probability for Chantal and an accordingly low probability for Sara. In the lower panel, the image shows a lady who is neither Chantal nor Sara, so the classifier predicts different sets of probabilities indicating a high uncertainty.



Bayesian Probabilistic Classifier

https://www.manning.com/books/probabilistic-deep-learning

SECA4002 – DEEP LEARNING NEURAL NETWORKS

Detailed Syllabus:

UNIT 2: DEEP NETWORKS

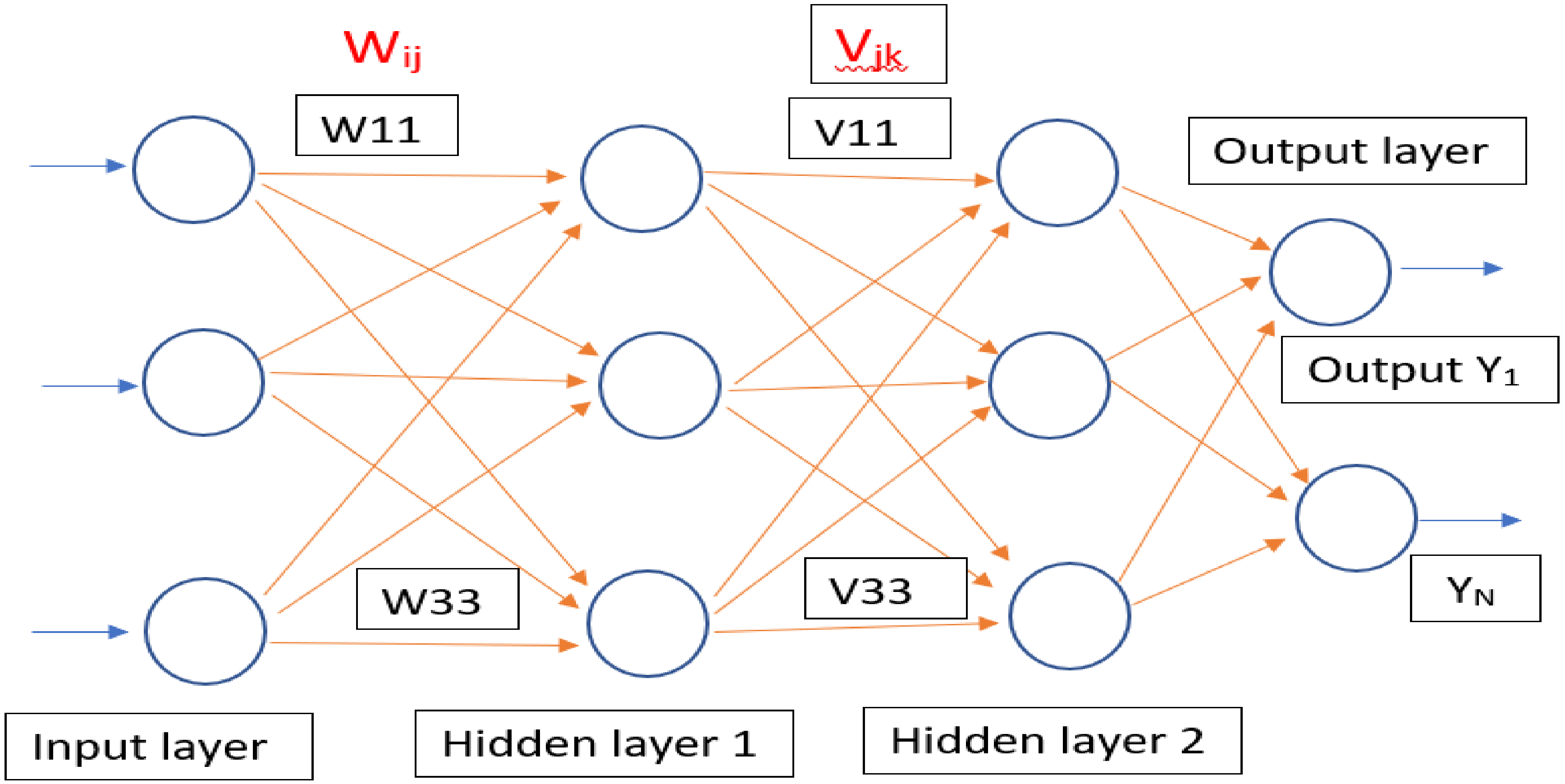
History of Deep Learning- A Probabilistic Theory of Deep Learning- Backpropagation and regularization, batch normalization- VC Dimension and Neural Nets-Deep Vs Shallow Networks Convolutional Networks- Generative Adversarial Networks (GAN), Semi-supervised Learning

Back Propagation Networks [BPN]

- Introduced by Rumelhart, Hinton, & Williams in 1986
- Multi layer Feedforward Network but error is back propagated, Hence the name Back Propagation Network (BPN)
- Uses Supervised Training process
- Systematic procedure for training the network is used
- For Error Detection and Correction Generalized Delta Law /Continuous Perceptron Law/ Gradient Descent Law is used
- Generalized Delta rule minimizes the mean squared error of the output calculated from the output
- Training by backpropagation involves three (3) stages

  1. Feedforward of input training pair
  2. Calculation and backpropagation of associated error
  3. Adjustments of weights
- Different variants of BPN are available for increasing the training speed of the network

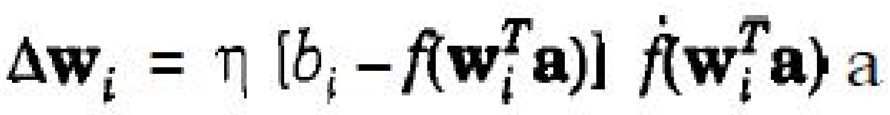
Back Propagation Networks [BPN]- Architecture



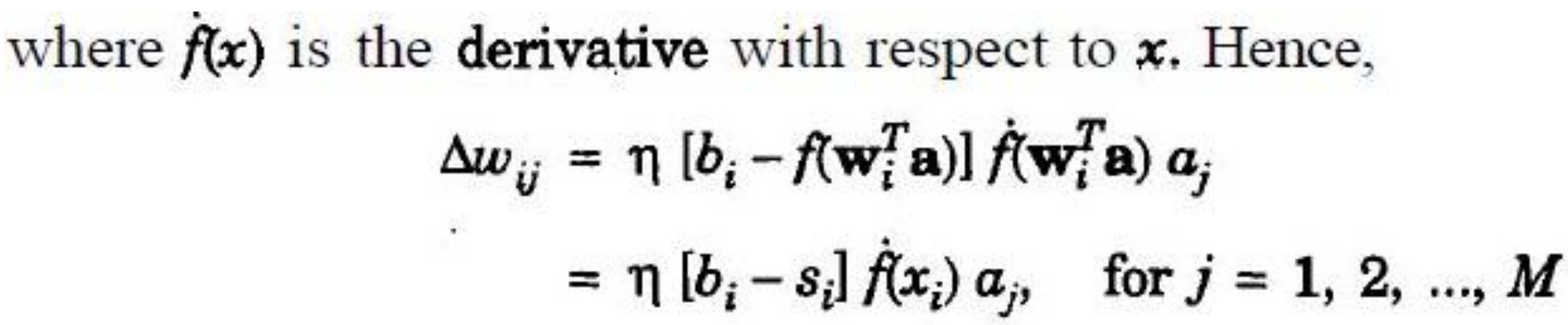
Weights between Input and Hidden Layer 1 is denoted as W_{IJ}
Weights between Hidden Layer 1 and Hidden Layer 2 is denoted as Vjk

Delta Learning Law

- Also known as Continuous perceptron Law ,Gradient Descent Law,
- Supervised Training Law



$$\Delta \mathbf{w}_i = \eta \, [b_i - f(\mathbf{w}_i^T \mathbf{a})] \, \dot{f}(\mathbf{w}_i^T \mathbf{a}) \, \mathbf{a}$$



where $\dot{f}(x)$ is the **derivative** with respect to x. Hence,

$$\Delta w_{ij} = \eta \, [b_i - f(\mathbf{w}_i^T \mathbf{a})] \, \dot{f}(\mathbf{w}_i^T \mathbf{a}) \, a_j$$

$$= \eta \, [b_i - s_i] \, \dot{f}(x_i) \, a_j, \quad \text{for } j = 1, 2, \ldots, M$$

Note:

- Valid only for Differential Output functions ,since weight change depends on the derivative of output function
- Supervised Training Law because change in weight is the difference between the Desired and Actual output (Error value)

Delta Learning Law

- Faster convergence rate when compared with Perceptron Law
- Extended version of Perceptron Training Law
- Used to calculate the local minima point
- Learning rate is taken in the range of 0.1 to 1

Limitations:

1. Network is slow for large data sets
2. Convergence time is more
3. Issues of Local minima
4. Temporal instability

Back Propagation Networks [BPN]- Algorithm

The training algorithm of BPN can be divided as

1. Initialization of Bias, Weights
2. Feedforward process
3. Back Propagation of Errors
4. Updating of weights & biases

Terms Used:

t – output Target Vector
X – Input Training vector
δ_k – Error at output unit Y_k
δ_j – Error at Hidden unit Z_j
α- Learning Rate
Voj- Bias on hidden unit j

Z_j – Hidden Unit j
Wok- Bias on Output unit k
Yk- Output unit K

Back Propagation Networks [BPN]- Algorithm

Algorithm:

I. Initialization of weights:

Step 1: Initialize the weights to small random values near zero

Step 2: While stop condition is false , Do steps 3 to 10

Step 3: For each training pair do steps 4 to 9

II. Feed forward of inputs

Step 4: Each input xi is received and forwarded to higher layers (next hidden)

Step 5: Hidden unit sums its weighted inputs as follows

$$Z_{inj} = W_{oj} + \Sigma x_i w_{ij}$$

Applying Activation function

$$Z_j = f(Z_{inj})$$

This value is passed to the output layer

Note: Bipolar Sigmoidal / Tanh function is used

Back Propagation Networks [BPN]- Algorithm

Algorithm (Continued)

Step 6: Output unit sums it’s weighted inputs

$$y_{ink}= V_{oj} + \Sigma\ Z_j V_{jk}$$

Applying Activation function

$$Y_k = f(y_{ink})$$

III. Backpropagation of Errors

Step 7: $\delta_k = (t_k – Y_k)f(y_{ink}\ )$

Step 8: $\delta_{inj} = \Sigma\ \delta_j V_{jk}$

IV. Updating of Weights & Biases

Step 8: Weight correction

$$\Delta w_{ij} = \alpha\delta_k Z_j$$

bias Correction

$$\Delta w_{oj} = \alpha\delta_k$$

Back Propagation Networks [BPN]- Algorithm

Algorithm (Continued)

IV. Updating of Weights & Biases

Step 9: continued:

New Weight is

$$W_{ij(new)} = W_{ij(old)} + \Delta w_{ij}$$

$$V_{jk(new)} = V_{jk(old)} + \Delta V_{jk}$$

New bias is

$$W_{oj(new)} = W_{oj(old)} + \Delta w_{oj}$$
$$V_{ok(new)} = V_{ok(old)} + \Delta V_{ok}$$

Step 10: Test for Stop Condition

Merits & Demerits of BPN Model

Merits:

- Has smooth effect on weight correction
- Computing time is less if weight’s are small
- 100 times faster than perceptron model
- Has a systematic weight updating procedure

Demerits:

- Learning phase requires intensive calculations
- Selection of number of Hidden layer neurons is an issue
- Selection of number of Hidden layers is also an issue
- Network gets trapped in Local Minima
- Temporal Instability
- Network Paralysis
- Training time is more for Complex problems

Regularization

- A fundamental problem in machine learning is how to make an algorithm that will perform well not just on the training data, but also on new inputs. Many strategies used in machine learning are explicitly designed to reduce the test error, possibly at the expense of increased training error. These strategies are known collectively as regularization

Definition: - "any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error."

- In the context of deep learning, most regularization strategies are based on regularizing estimators.
- Regularization of an estimator works by trading increased bias for reduced variance.

An effective regularizer is one that makes a profitable trade, reducing variance significantly while not overly increasing the bias.

Regularization - Parameter Norm Penalties

- ❑ Many regularization approaches are based on limiting the capacity of models, such as neural networks, linear regression, or logistic regression, by adding a parameter norm penalty $\Omega(\theta)$ to the objective function J. We denote the regularized objective function by J˜

$$J\tilde{}(\theta; X, y) = J(\theta; X, y) + \alpha\Omega(\theta)$$

  - ➢ where $\alpha \in [0, \infty)$ is a hyperparameter that weights the relative contribution of the norm penalty term, Ω, relative to the standard objective function J. Setting α to 0 results in no regularization.
  - ➢ Larger values of α correspond to more regularization

- ❖ The parameter norm penalty Ω that penalizes only the weights of the affine transformation at each layer and leaves the biases unregularized

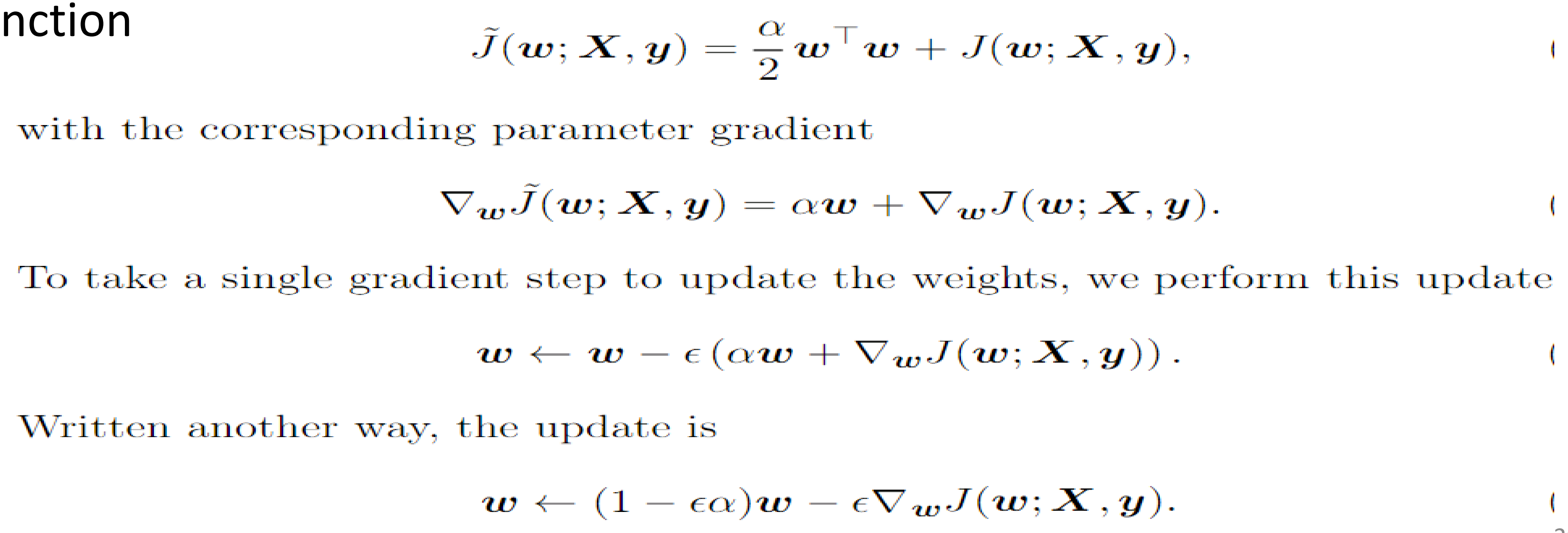
L^2 Parameter Regularization

One of the simplest and most common kind of parameter norm penalty is L^2 parameter & it's also called commonly as **weight decay**

This regularization strategy drives the weights closer to the origin by adding a regularization term Ω(θ)= ½||w||½

L2 regularization is also known as ridge regression or Tikhonov regularization.

To simplify, we assume no bias parameter, so θ is just w. Such a model has the following total objective function



$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \frac{\alpha}{2} \boldsymbol{w}^\top \boldsymbol{w} + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}),$$

with the corresponding parameter gradient

$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \boldsymbol{w} + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}).$$

To take a single gradient step to update the weights, we perform this update

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \epsilon \left(\alpha \boldsymbol{w} + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) \right).$$

Written another way, the update is

$$\boldsymbol{w} \leftarrow (1 - \epsilon\alpha) \boldsymbol{w} - \epsilon \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}).$$

L^2 Parameter Regularization

- We can see that the addition of the weight decay term has modified the learning rule to multiplicatively shrink the weight vector by a constant factor on each step, just before performing the usual gradient update. This describes what happens in a single step.
- The approximation ^J is Given by

$$\hat{J}(\boldsymbol{\theta}) = J(\boldsymbol{w}^*) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^*)^\top \boldsymbol{H}(\boldsymbol{w} - \boldsymbol{w}^*),$$

Where H is the Hessian matrix of J with respect to w evaluated at w∗.

The minimum of ˆJ occurs where its gradient ∇wˆJ(w) = H(w − w*) is equal to ‘0’

To study the effect of weight decay,

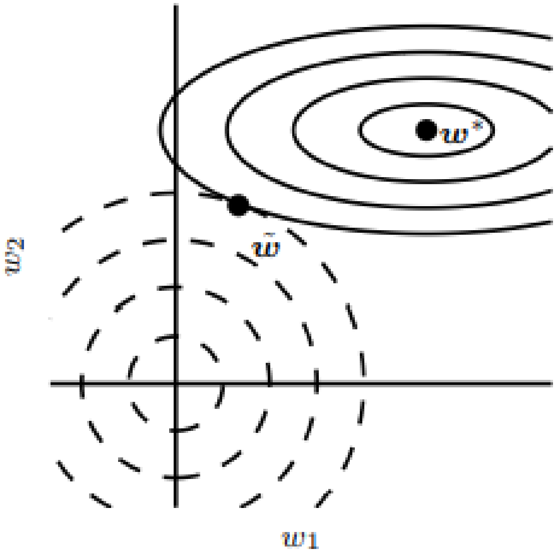
$$\alpha\tilde{\boldsymbol{w}} + \boldsymbol{H}(\tilde{\boldsymbol{w}} - \boldsymbol{w}^*) = 0$$
$$(\boldsymbol{H} + \alpha\boldsymbol{I})\tilde{\boldsymbol{w}} = \boldsymbol{H}\boldsymbol{w}^*$$
$$\tilde{\boldsymbol{w}} = (\boldsymbol{H} + \alpha\boldsymbol{I})^{-1}\boldsymbol{H}\boldsymbol{w}^*$$

As α approaches 0, the regularized solution ˜w approaches w*. But what happens as α grows? Because H is real and symmetric, we can decompose it into a diagonal matrix Λ and an orthonormal basis of eigenvectors, Q, such that $H = Q\Lambda Q^T$

L^2 Parameter Regularization

❖ Applying Decomposition to the above equation, We Obtain

$$\begin{aligned}\tilde{\boldsymbol{w}} &= (\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top + \alpha\boldsymbol{I})^{-1}\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top\boldsymbol{w}^* \\ &= \left[\boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha\boldsymbol{I})\boldsymbol{Q}^\top\right]^{-1}\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top\boldsymbol{w}^* \\ &= \boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha\boldsymbol{I})^{-1}\boldsymbol{\Lambda}\boldsymbol{Q}^\top\boldsymbol{w}^*.\end{aligned}$$



The solid ellipses represent contours of equal value of the unregularized objective. The dotted circles represent contours of equal value of the L 2 regularizer. At the point w˜, these competing objectives reach an equilibrium. In the first dimension, the eigenvalue of the Hessian of J is small. The objective function does not increase much when moving horizontally away from w∗ . Because the objective function does not express a strong preference along this direction, the regularizer has a strong effect on this axis. The regularizer pulls w1 close to zero. In the second dimension, the objective function is very sensitive to movements away from w∗ . The corresponding eigenvalue is large, indicating high curvature. As a result, weight decay affects the position of w2 relatively little.

L^1 Parameter Regularization

While L 2 weight decay is the most common form of weight decay, there are other ways to penalize the size of the model parameters. Another option is to use L 1 regularization.

L 1 regularization on the model parameter w is defined as

$$\Omega(\boldsymbol{\theta}) = ||\boldsymbol{w}||_1 = \sum_i |w_i|,$$

- ➢ as the sum of absolute values of the individual parameters.

L 1 weight decay controls the strength of the regularization by scaling the penalty Ω using a positive hyperparameter α. Thus, the regularized objective function J˜(w; X, y) is given by

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha ||\boldsymbol{w}||_1 + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}),$$

with the corresponding gradient as

$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \alpha \operatorname{sign}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}} J(\boldsymbol{X}, \boldsymbol{y}; \boldsymbol{w}), \quad \longrightarrow \quad \text{Eq-1}$$

L^1 Parameter Regularization

By inspecting equation 1, we can see immediately that the effect of L 1 regularization is quite different from that of L 2 regularization. Specifically, we can see that the regularization contribution to the gradient no longer scales linearly with each wi ; instead it is a constant factor with a sign equal to sign(wi).

Quadratic approximation of the L 1 regularized objective function decomposes into a sum over the parameters

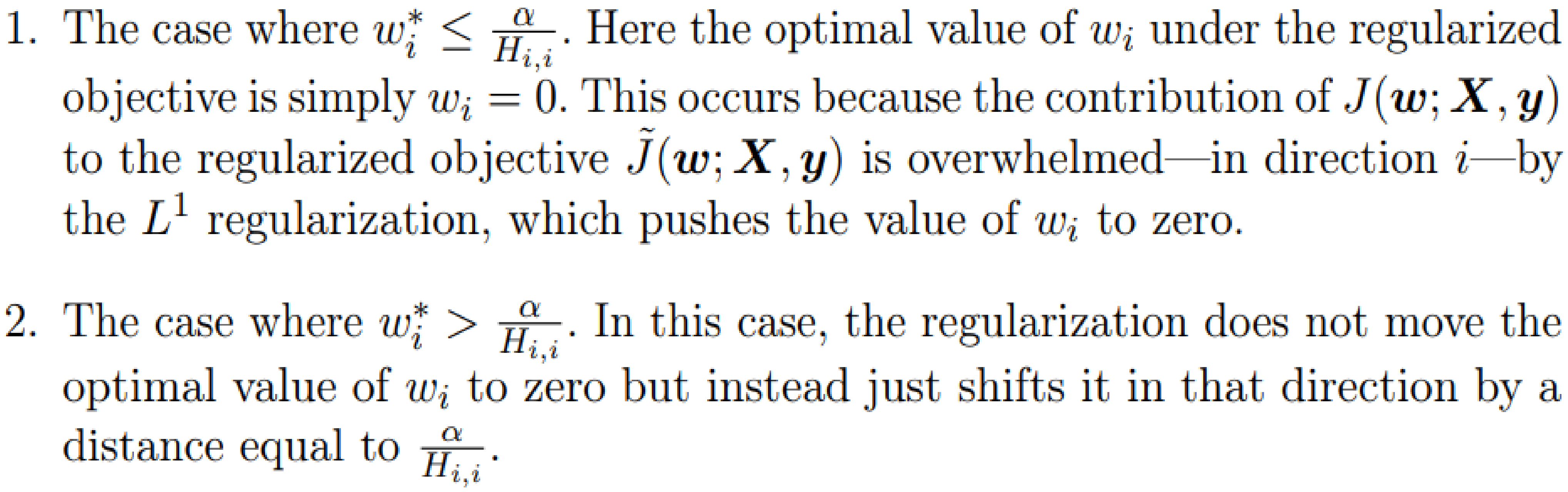
$$\hat{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{w}^*; \boldsymbol{X}, \boldsymbol{y}) + \sum_i \left[\frac{1}{2} H_{i,i} (\boldsymbol{w}_i - \boldsymbol{w}_i^*)^2 + \alpha |w_i| \right].$$

The problem of minimizing this approximate cost function has an analytical solution with the following form:

$$w_i = \operatorname{sign}(w_i^*) \max \left\{ |w_i^*| - \frac{\alpha}{H_{i,i}}, 0 \right\}.$$

L^1 Parameter Regularization

Consider the situation where w ∗ i > 0 for all i. There are two possible outcomes:



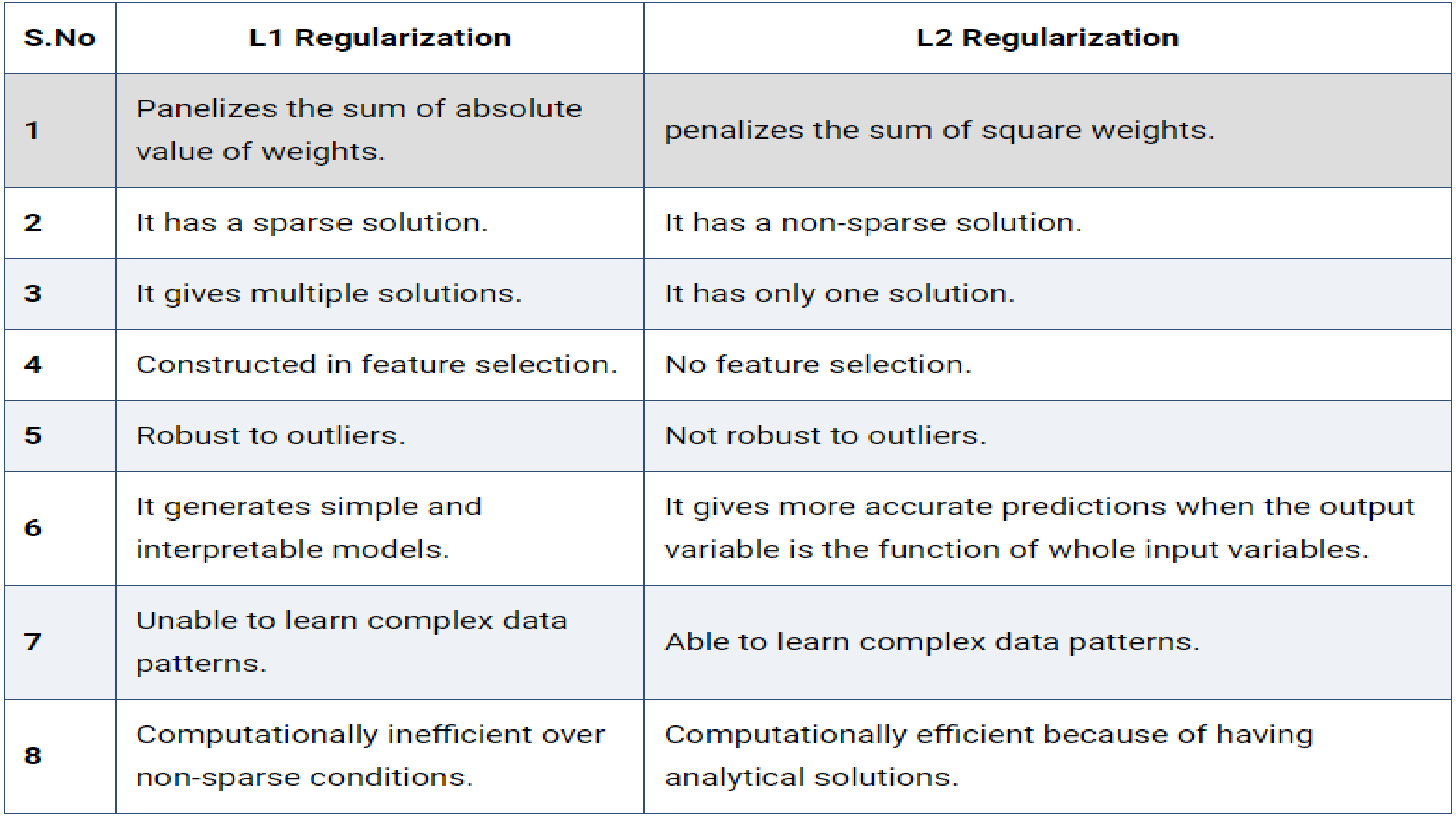
1. The case where $w_i^* \le \frac{\alpha}{H_{i,i}}$. Here the optimal value of w_i under the regularized objective is simply $w_i = 0$. This occurs because the contribution of $J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$ to the regularized objective $\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$ is overwhelmed—in direction i—by the L^1 regularization, which pushes the value of w_i to zero.

2. The case where $w_i^* > \frac{\alpha}{H_{i,i}}$. In this case, the regularization does not move the optimal value of w_i to zero but instead just shifts it in that direction by a distance equal to $\frac{\alpha}{H_{i,i}}$.

Difference between L^1 & L^2 Parameter Regularization

- L1 regularization attempts to estimate the median of data, L2 regularization makes estimation for the mean of the data in order to evade overfitting.
- L1 regularization can add the penalty term in cost function. But L2 regularization appends the squared value of weights in the cost function.
- L1 regularization can be helpful in features selection by eradicating the unimportant features, whereas, L2 regularization is not recommended for feature selection
- L1 doesn't have a closed form solution since it includes an absolute value and it is a non-differentiable function, while L2 has a solution in closed form as it's a square of a weight

Difference between L^1 & L^2 Parameter Regularization



S.No	L1 Regularization	L2 Regularization
1	Panelizes the sum of absolute value of weights.	penalizes the sum of square weights.
2	It has a sparse solution.	It has a non-sparse solution.
3	It gives multiple solutions.	It has only one solution.
4	Constructed in feature selection.	No feature selection.
5	Robust to outliers.	Not robust to outliers.
6	It generates simple and interpretable models.	It gives more accurate predictions when the output variable is the function of whole input variables.
7	Unable to learn complex data patterns.	Able to learn complex data patterns.
8	Computationally inefficient over non-sparse conditions.	Computationally efficient because of having analytical solutions.

SECA4002 – DEEP LEARNING NEURAL NETWORKS

Detailed Syllabus:

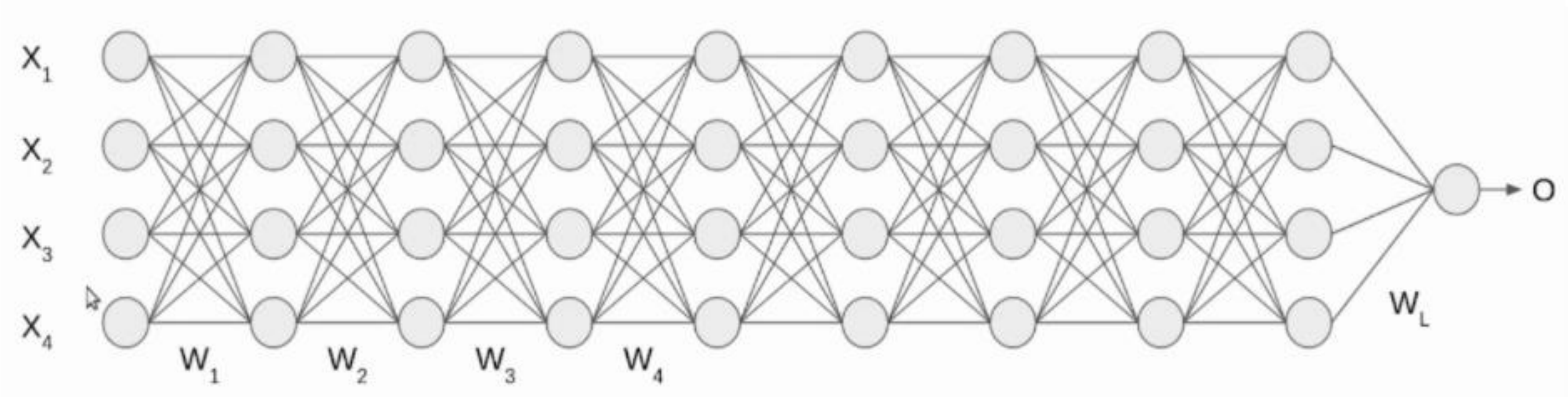
UNIT 2: DEEP NETWORKS

History of Deep Learning- A Probabilistic Theory of Deep Learning- Backpropagation and regularization, Batch normalization- VC Dimension and Neural Nets-Deep Vs Shallow Networks Convolutional Networks- Generative Adversarial Networks (GAN), Semi-supervised Learning

Batch Normalization

- It is a method of adaptive reparameterization, motivated by the difficulty of training very deep models.
- In Deep networks, the weights are updated for each layer. So the output will no longer be on the same scale as the input (even though input is normalized).
- Normalization - is a data pre-processing tool used to bring the numerical data to a common scale without distorting its shape.
- when we input the data to a machine or deep learning algorithm we tend to change the values to a balanced scale because, we ensure that our model can generalize appropriately.(Normalization is used to bring the input into a balanced scale/ Range)

Let's understand this through an example, we have a deep neural network as shown in the following image.



L = Number of layers

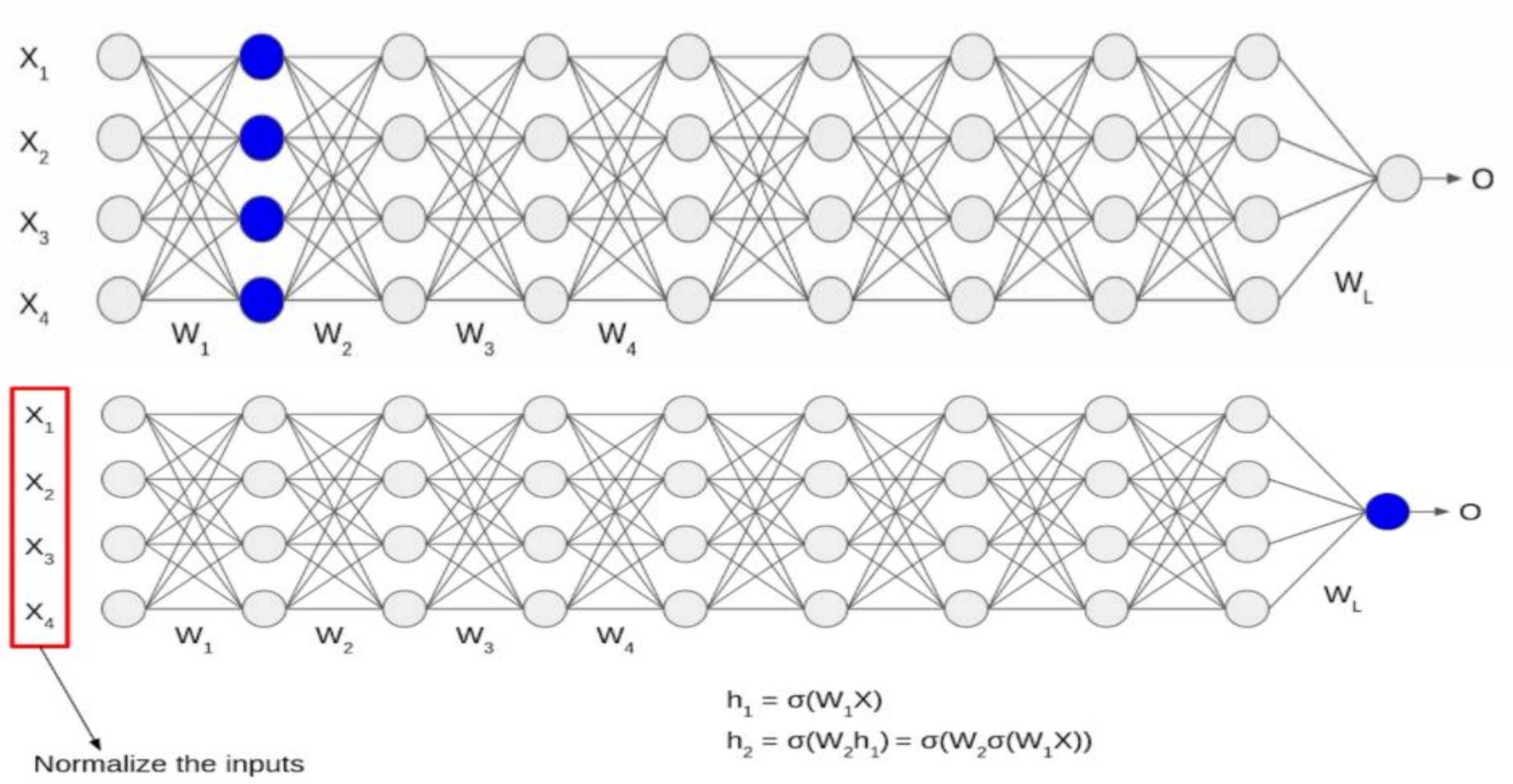
Bias = 0

Activation Function: Sigmoid

Image Source: https://www.analyticsvidhya.com/blog/2021/03/introduction-to-batch-normalization/

Batch Normalization

Initially, our inputs X1, X2, X3, X4 are in normalized form as they are coming from the pre-processing stage. When the input passes through the first layer, it transforms, as a sigmoid function applied over the dot product of input X and the weight matrix W.



$$h_1 = \sigma(W_1X)$$

$$h_1 = \sigma(W_1X)$$
$$h_2 = \sigma(W_2h_1) = \sigma(W_2\sigma(W_1X))$$

$$O = \sigma(W_Lh_{L-1})$$

Image Source: https://www.analyticsvidhya.com/blog/2021/03/introduction-to-batch-normalization/

Batch Normalization

Even though the input X was normalized but the output is no longer on the same scale.
The data passes through multiple layers of network with multiple times(sigmoidal) activation functions are applied, which leads to an internal co-variate shift in the data.

This motivates us to move towards Batch Normalization

Normalization is the process of altering the input data to have mean as zero and standard deviation value as one

Procedure to do Batch Normalization:

(1) Consider the batch input from layer h, for this layer we need to calculate the mean of this hidden activation.

$$\mu = \frac{1}{m} \sum h_i$$

Where n - number of neurons in the hidden layer h

Image Source: https://www.analyticsvidhya.com/blog/2021/03/introduction-to-batch-normalization/

Batch Normalization (BN)

(2) After calculating the mean the next step is to calculate the standard deviation of the hidden activations.

$$\sigma = \left[\frac{1}{m} \sum (h_i - \mu)^2 \right]^{1/2}$$

(3)cNow we normalize the hidden activations using these Mean & Standard Deviation values.
To do this, we subtract the mean from each input and divide the whole value with the sum of standard deviation and the smoothing term (ε).

$$h_{i(norm)} = \frac{(h_i - \mu)}{\sigma + \varepsilon}$$

(4) As the final stage, the re-scaling and offsetting of the input is performed. Here two components of the BN algorithm is used, γ(gamma) and β (beta). These parameters are used for re-scaling (γ) and shifting(β) the vector contains values from the previous operations.

$$h_i = \gamma \, h_{i(norm)} + \beta$$

These two parameters are learnable parameters, Hence during the training of neural network, the optimal values of γ and β are obtained and used. Hence we get the accurate normalization of each batch.

All equations are adopted from:https://www.analyticsvidhya.com/blog/2021/03/introduction-to-batch-normalization/

SECA4002 – DEEP LEARNING NEURAL NETWORKS

Detailed Syllabus:

UNIT 2: DEEP NETWORKS

History of Deep Learning- A Probabilistic Theory of Deep Learning- Backpropagation and regularization, Batch normalization- VC Dimension and Neural Nets-Deep Vs Shallow Networks Convolutional Networks- Generative Adversarial Networks (GAN), Semi-supervised Learning

Shallow Networks

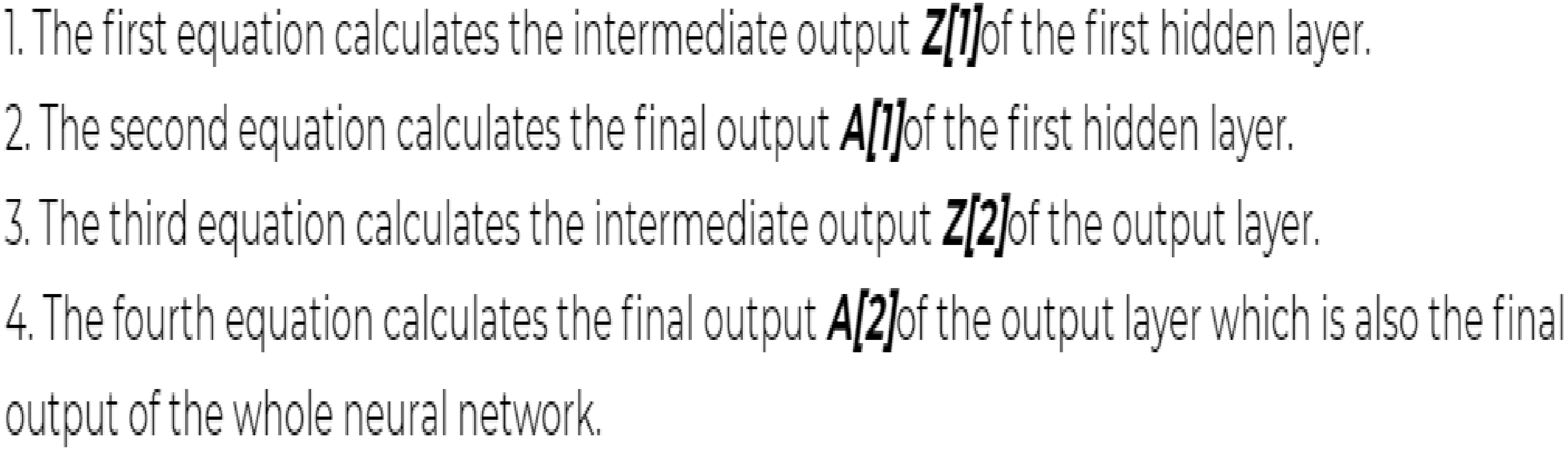
Shallow neural networks give us basic idea about deep neural network which consist of only 1 or 2 hidden layers. Understanding a shallow neural network gives us an understanding into what exactly is going on inside a deep neural network A neural network is built using various hidden layers. Now that we know the computations that occur in a particular layer, let us understand how the whole neural network computes the output for a given input ***X***. These can also be called the ***forward-propagation*** equations.

$$Z^{[1]} = W^{[1]T}X + b^{[1]}$$

$$A^{[1]} = \sigma\left(Z^{[1]}\right)$$

$$Z^{[2]} = W^{[2]T}A^{[1]} + b^{[2]}$$

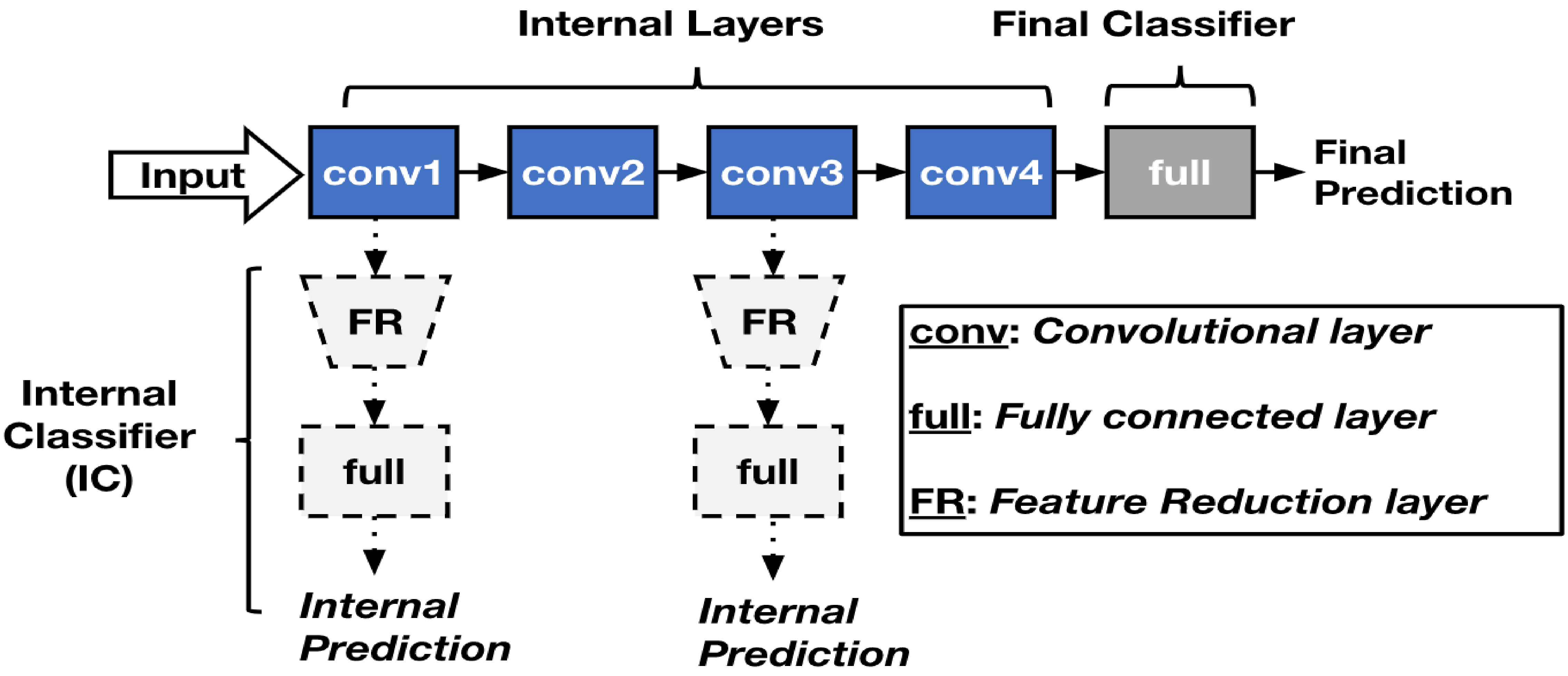
$$\hat{y} = A^{[2]} = \sigma\left(Z^{[2]}\right)$$



1. The first equation calculates the intermediate output ***Z[1]***of the first hidden layer.
2. The second equation calculates the final output ***A[1]***of the first hidden layer.
3. The third equation calculates the intermediate output ***Z[2]***of the output layer.
4. The fourth equation calculates the final output ***A[2]***of the output layer which is also the final output of the whole neural network.

Shallow Networks

Shallow-Deep Networks: A Generic Modification to Deep Neural Networks



Shallow Networks

Difference Between a Shallow Net & Deep Learning Net:

Sl.No	Shallow Net’s	Deep Learning Net’s
1	One Hidden layer(or very less no. of Hidden Layers)	Deep Net’s has many layers of Hidden layers with more no. of neurons in each layers
2	Takes input only as VECTORS	DL can have raw data like image, text as inputs
3	Shallow net’s needs more parameters to have better fit	DL can fit functions better with less parameters than a shallow network
4	Shallow networks with one Hidden layer (same no of neurons as DL) cannot place complex functions over the input space	DL can compactly express highly complex functions over input space
5	The number of units in a shallow network grows exponentially with task complexity.	DL don’t need to increase it size(neurons) for complex problems
6	Shallow network is more difficult to train with our current algorithms (e.g. it has issues of local minima etc)	Training in DL is easy and no issue of local minima in DL

VC Dimensions

Vapnik–Chervonenkis (VC) dimension is a measure of the capacity (complexity, expressive power, richness, or flexibility) of a set of functions that can be learned by a statistical binary classification algorithm

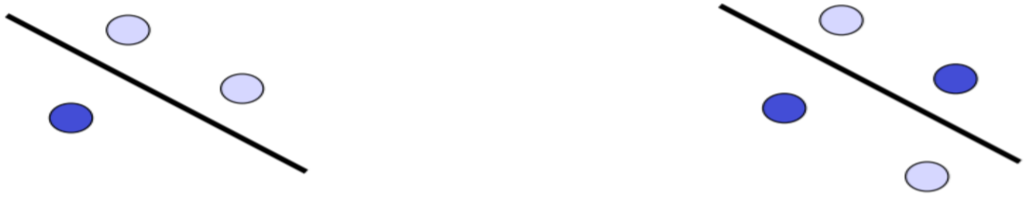
Definition: It is defined as the cardinality of the largest set of points that the algorithm can shatter, which means the algorithm can always learn a perfect classifier for any labeling of that many data points

Source: https://en.wikipedia.org/wiki/Vapnik%E2%80%93Chervonenkis_dimension (Wikipedia)

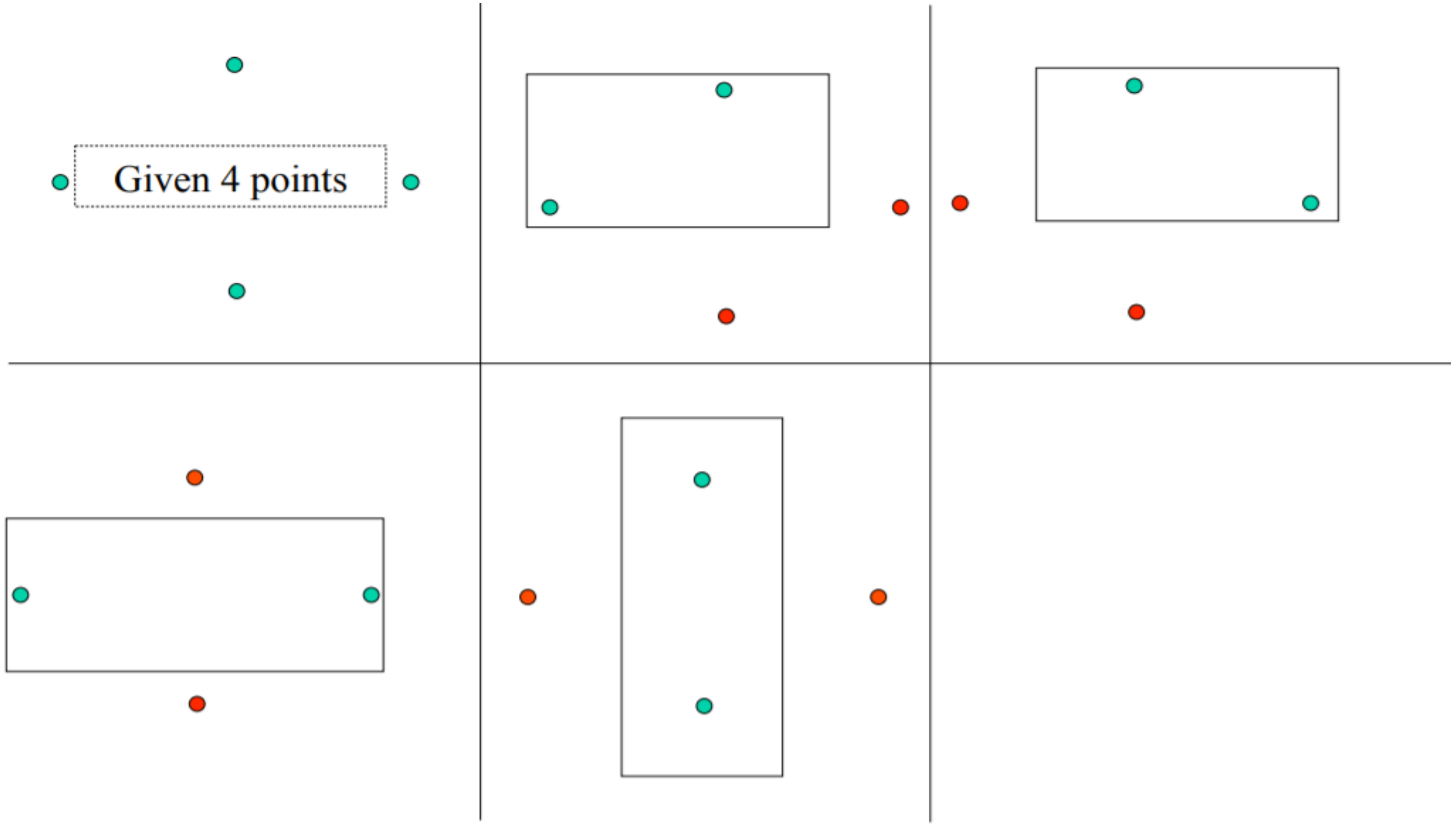
- The maximum number of datapoints that can be separated (i.e., grouped) in *all* possible ways
- More powerful representations are able to shatter larger sets of datapoints. These have higher VC dimension.
- Less powerful representations can only shatter smaller sets of data points. These then have lower VC dimension.

The VC dimension measures the capacity of a binary classifier.

VC Dimensions



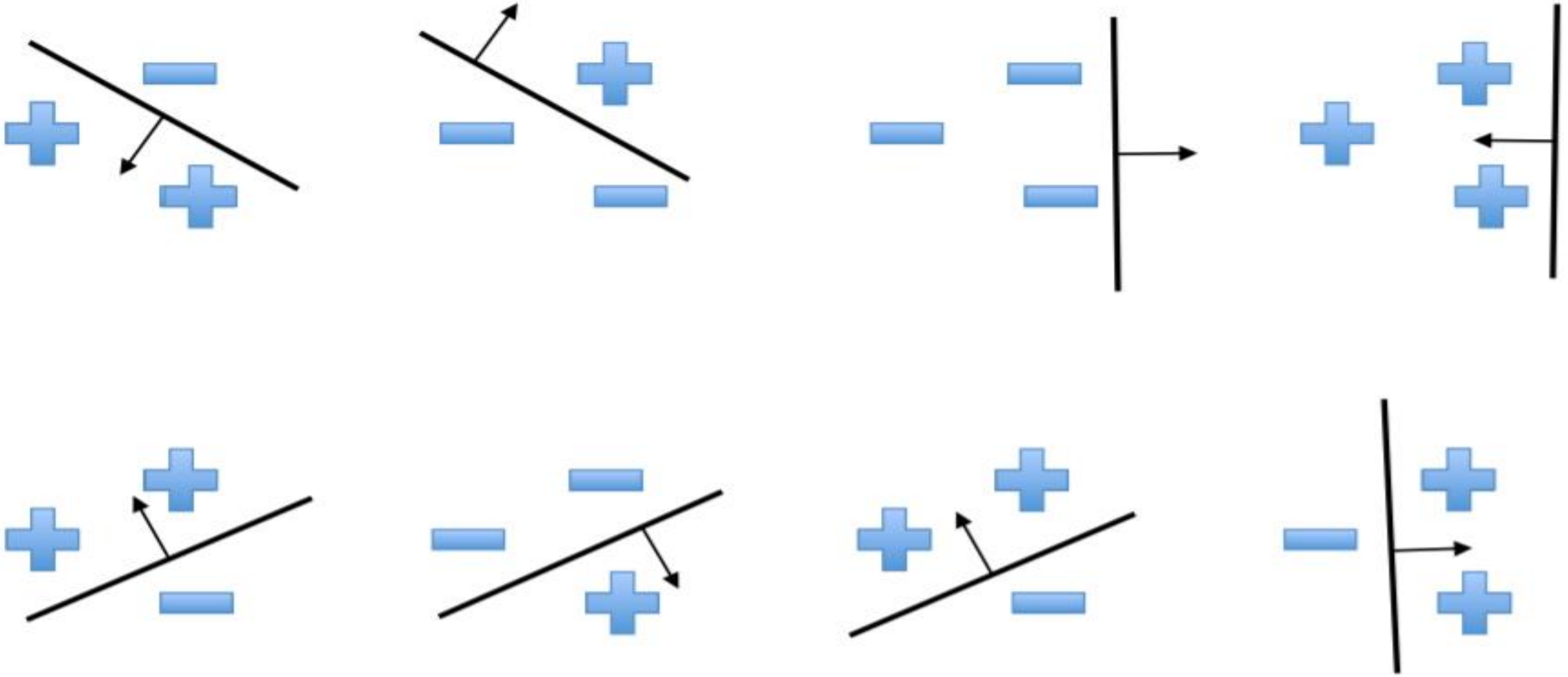
For n=2 then we can have 2^n possible classifications



2 points “+” and 2 points “-” ⇒ we can define a rectangle which includes the 2 “+” and excludes the 2 “-”.

VC Dimensions

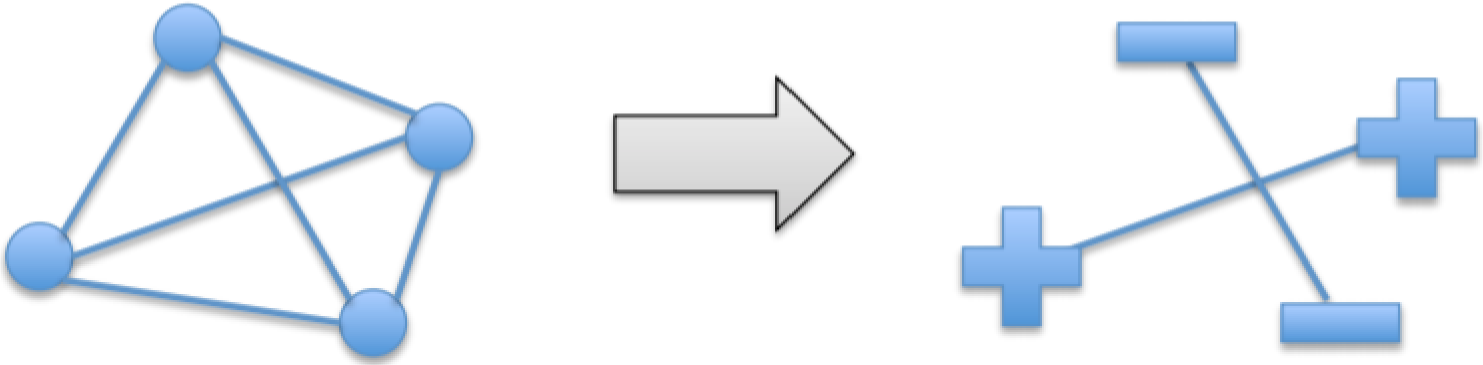
- Suppose our model class is a hyperplane
- Consider all labelings over three points in $\mathbb{R}^2$



- In $\mathbb{R}^2$, we can find a plane (i.e., a line) to capture any labeling of 3 points. A 2D hyperplane shatters 3 points

VC Dimensions

- But, a 2D hyperplane cannot deal with some labelings of four points:



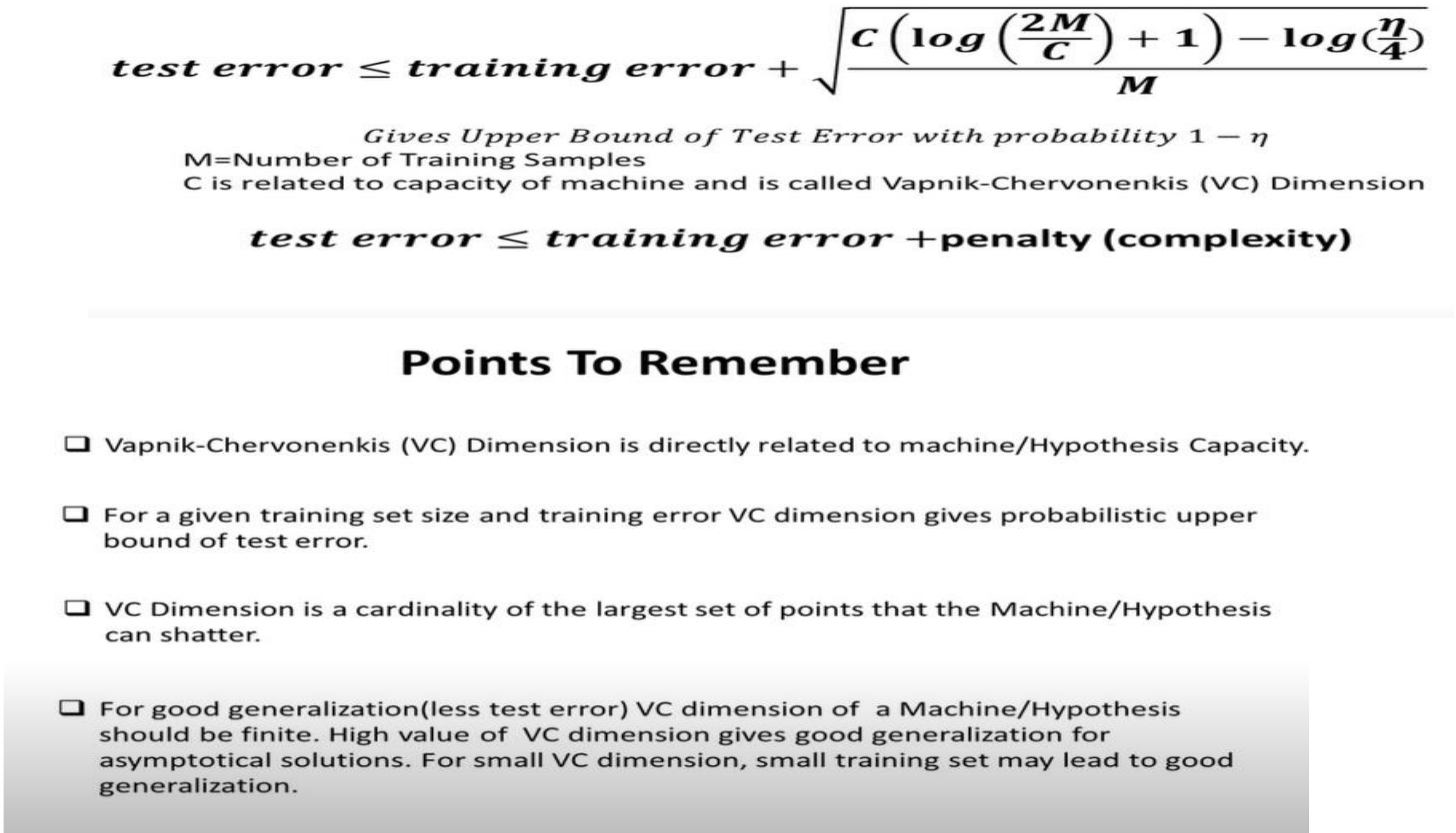
Connect all pairs of points; two lines will always cross

Can’t separate points if the pairs that cross are the same class

- Therefore, a 2D hyperplane cannot shatter 4 points

VC Dimensions

Upper Bound:



$$test\ error \leq training\ error + \sqrt{\frac{C\left(\log\left(\frac{2M}{C}\right)+1\right)-\log\left(\frac{\eta}{4}\right)}{M}}$$

Gives Upper Bound of Test Error with probability $1-\eta$

M=Number of Training Samples

C is related to capacity of machine and is called Vapnik-Chervonenkis (VC) Dimension

$$test\ error \leq training\ error + \textbf{penalty (complexity)}$$

Points To Remember

- ❑ Vapnik-Chervonenkis (VC) Dimension is directly related to machine/Hypothesis Capacity.
- ❑ For a given training set size and training error VC dimension gives probabilistic upper bound of test error.
- ❑ VC Dimension is a cardinality of the largest set of points that the Machine/Hypothesis can shatter.
- ❑ For good generalization(less test error) VC dimension of a Machine/Hypothesis should be finite. High value of VC dimension gives good generalization for asymptotical solutions. For small VC dimension, small training set may lead to good generalization.

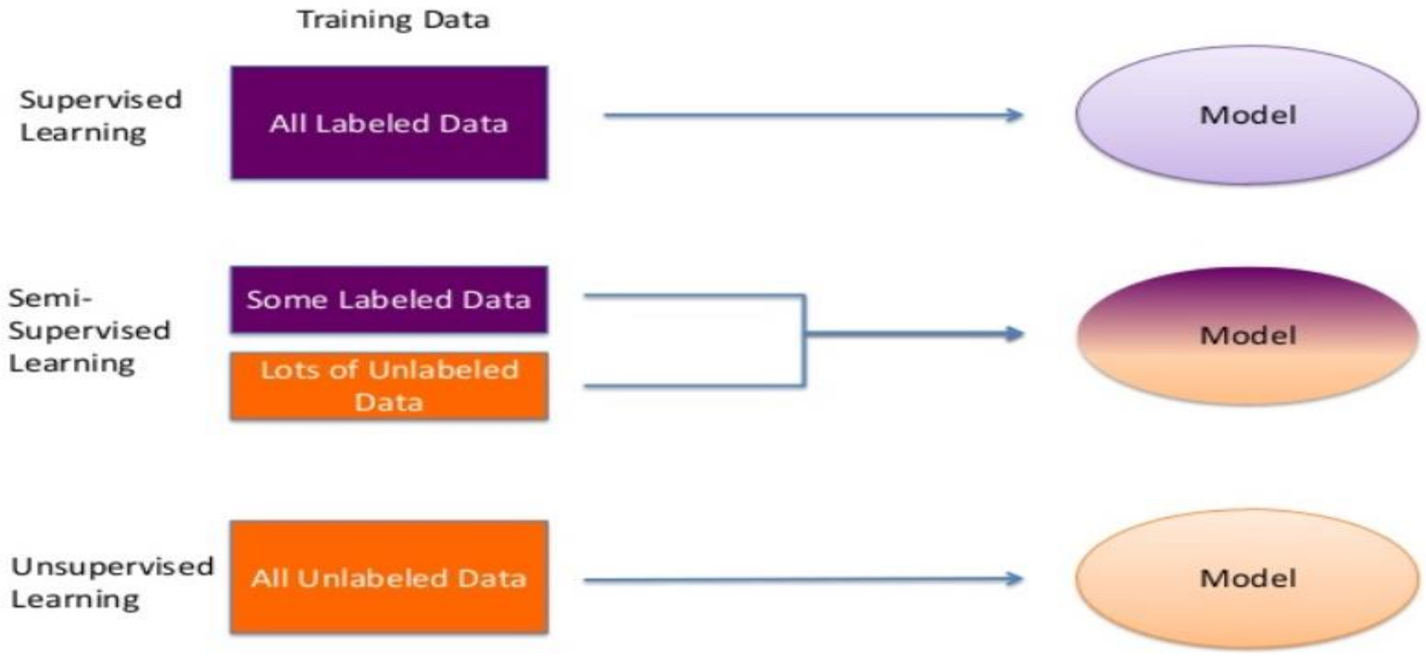
Semi Supervised Learning

- ❖ Semi-supervised learning is a type of machine learning

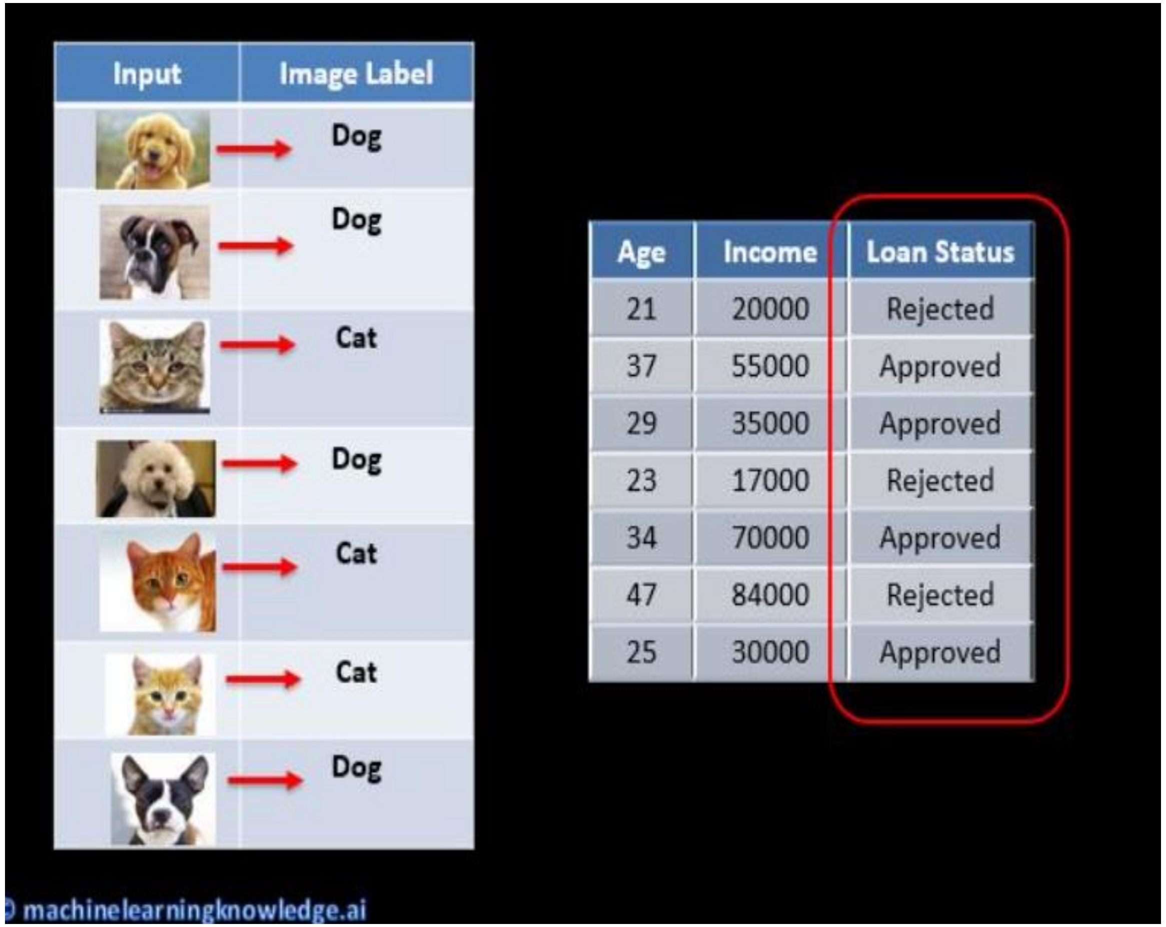
❑ It refers to a learning problem (and algorithms designed for the learning problem) that involves a small portion of labeled examples and a large number of unlabeled examples from which a model must learn and make predictions on new examples.

The programmer will cluster similar data using an unsupervised learning algorithm and then use the existing labeled data to label the rest of the unlabelled data

❑ Semi-Supervised learning where a teacher teaches a few concepts in class and gives questions as homework which are based on similar concepts.



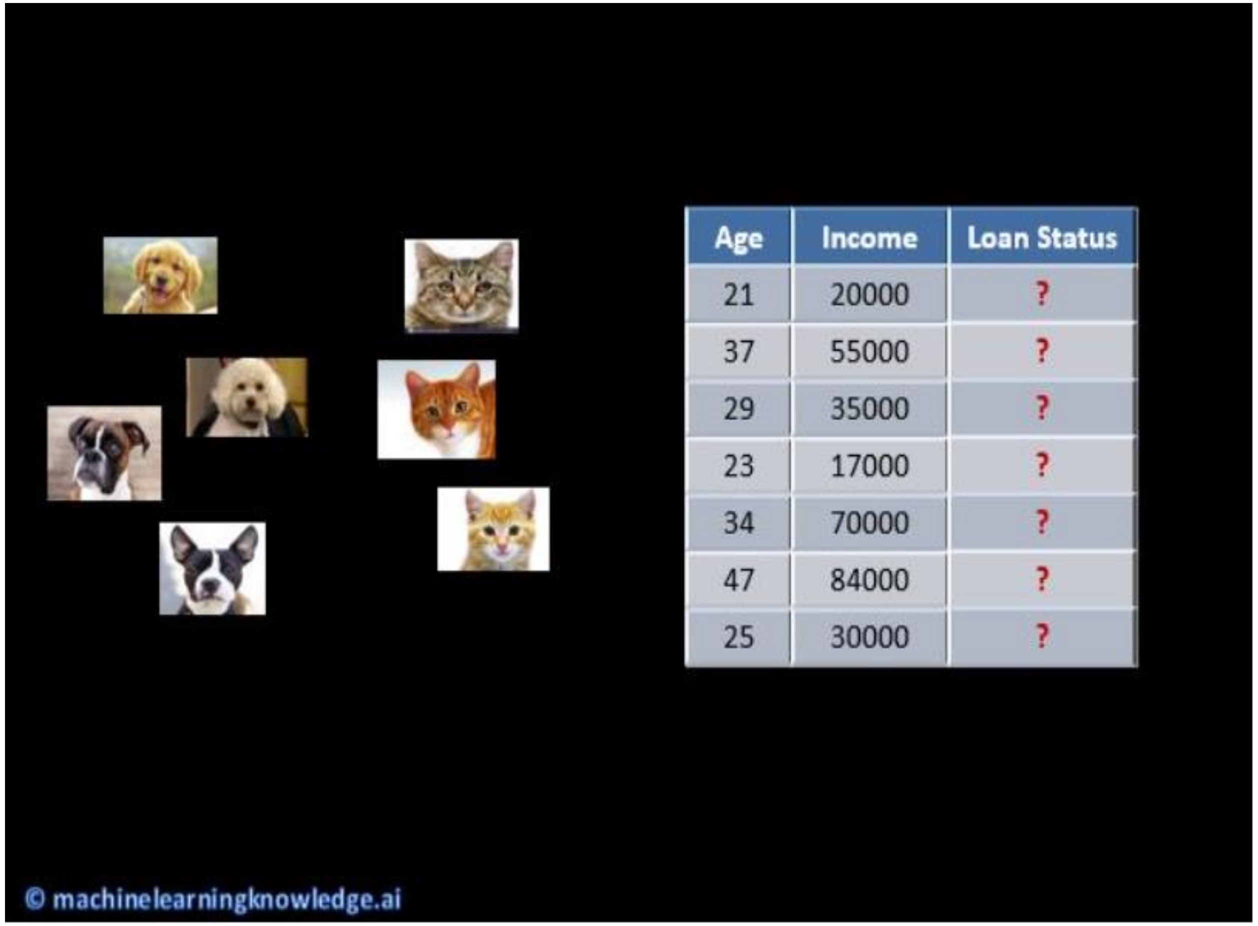
Semi Supervised Learning



Input	Image Label
	Dog
	Dog
	Cat
	Dog
	Cat
	Cat
	Dog

Age	Income	Loan Status
21	20000	Rejected
37	55000	Approved
29	35000	Approved
23	17000	Rejected
34	70000	Approved
47	84000	Rejected
25	30000	Approved

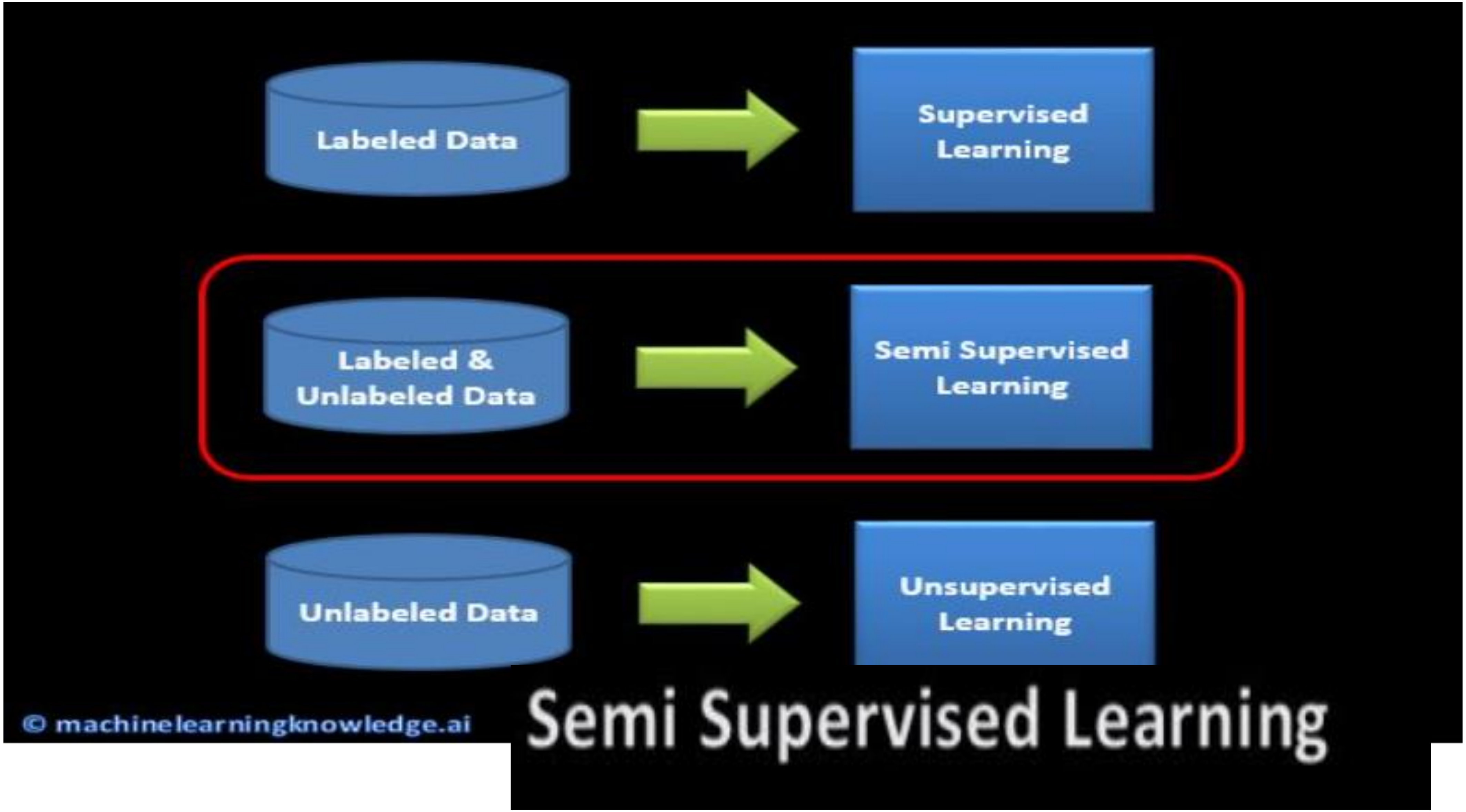
Example for Labelled data



Age	Income	Loan Status
21	20000	?
37	55000	?
29	35000	?
23	17000	?
34	70000	?
47	84000	?
25	30000	?

Example for Un-Labelled data

Semi Supervised Learning

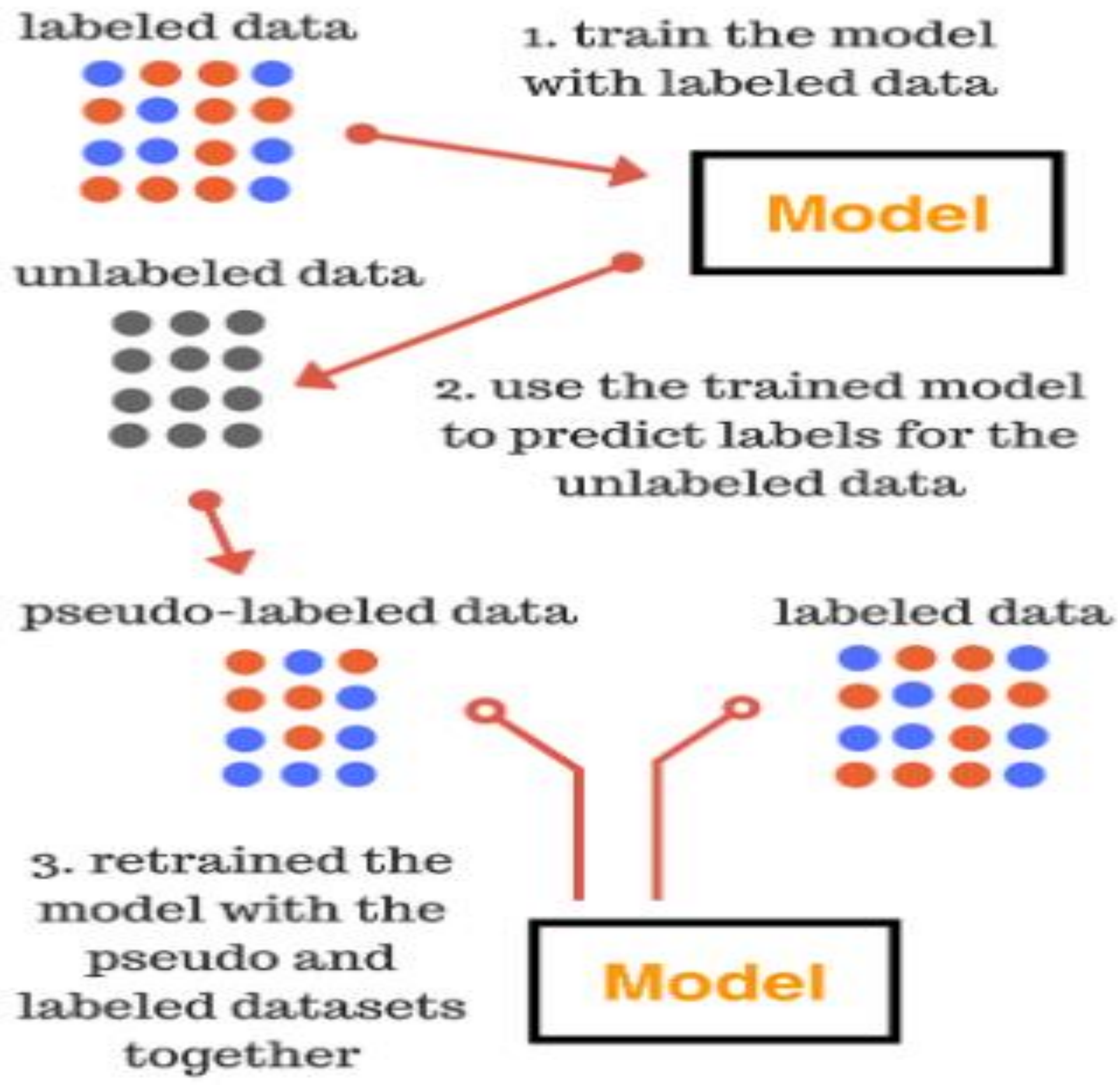


© machinelearningknowledge.ai

Semi Supervised Learning

Image source: https://machinelearningknowledge.ai/semi-supervised-learning-a-gentle-introduction-for-beginners/

Semi Supervised Learning


labeled data
1. train the model with labeled data
Model
unlabeled data
2. use the trained model to predict labels for the unlabeled data
pseudo-labeled data
labeled data
3. retrained the model with the pseudo and labeled datasets together
Model

Concept of Pseudo-Labelling : A Semi-Supervised learning technique

Semi Supervised Learning

Practical applications of Semi-Supervised Learning –

1. **Speech Analysis:** Since labeling of audio files is a very intensive task, Semi-Supervised learning is a very natural approach to solve this problem.
2. **Internet Content Classification:** Labeling each webpage is an impractical and unfeasible process and thus uses Semi-Supervised learning algorithms. Even the Google search algorithm uses a variant of Semi-Supervised learning to rank the relevance of a webpage for a given query.
3. **Protein Sequence Classification:** Since DNA strands are typically very large in size, the rise of Semi-Supervised learning has been imminent in this field.

Drawbacks:

Semi supervised learning when both labeled and unlabeled data is present does not guarantee a model with better performance.

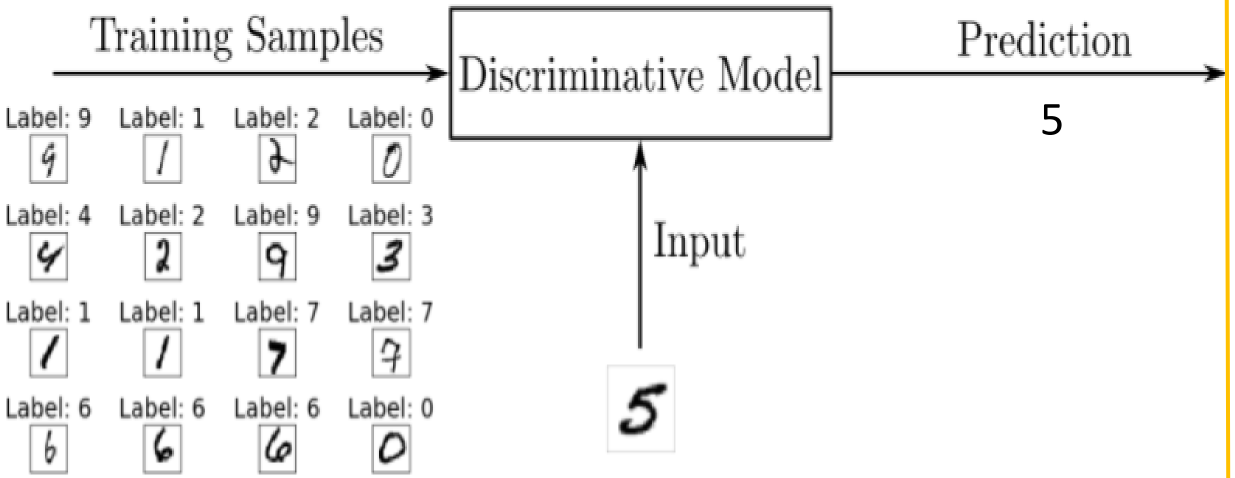
Difference between Semi-supervised and Reinforcement Learning:

Reinforcement learning is different from semi-supervised learning, as it works with rewards and feedback. ***Reinforcement learning aims to maximize the rewards by their hit and trial actions, whereas in semi-supervised learning, we train the model with a less labeled dataset.***

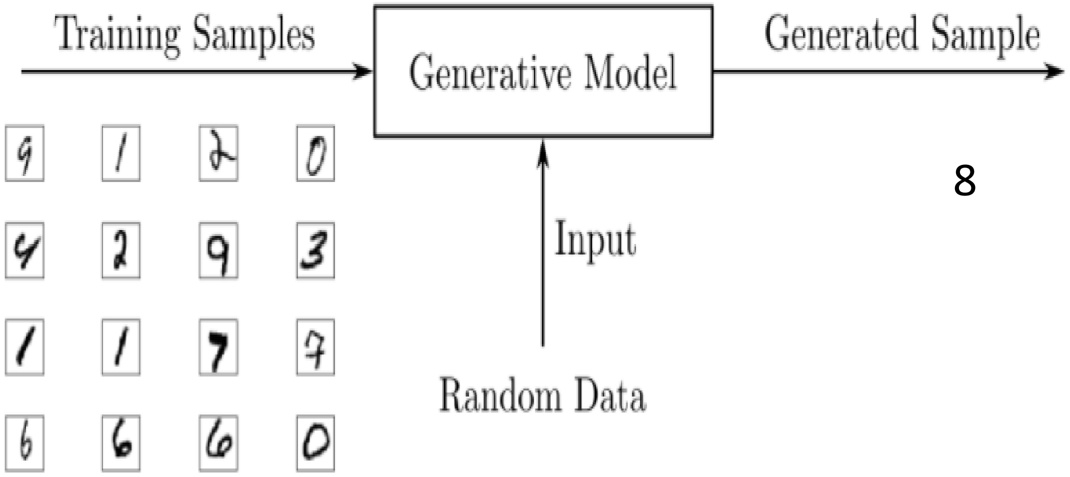
Generative Adversarial Networks (GAN's)

- ❑ Generative Adversarial Networks, are an approach to generative modeling using deep learning methods, such as convolutional neural networks
- ❑ Generative modeling is an unsupervised learning task in machine learning that involves automatically discovering and learning the regularities or patterns in input data
- ❑ GANs consist of two neural networks, one trained to generate data and the other trained to distinguish fake data from real data (hence the "adversarial" nature of the model)

Generative adversarial networks are machine learning systems that can learn to mimic a given distribution of data. They were first proposed in a 2014 NeurIPS paper by deep learning expert Ian Goodfellow and his colleagues.



Discriminative models are those used for most supervised **classification** or **regression** problems



Generative models are often used with unlabeled datasets and can be seen as a form of unsupervised learning.

Generative Adversarial Networks (GAN's)

Analogy

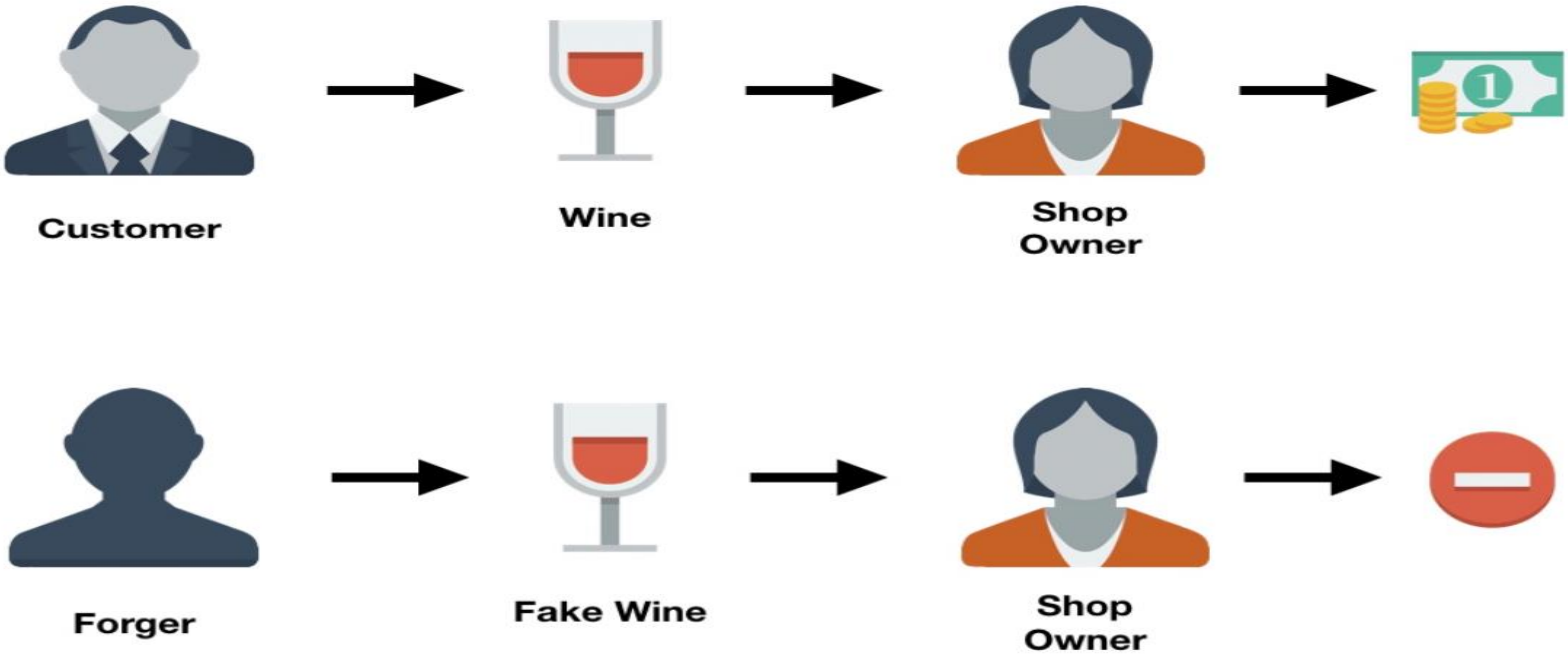


Image Source: https://www.datacamp.com/community/tutorials/generative-adversarial-networks

Components of a GAN

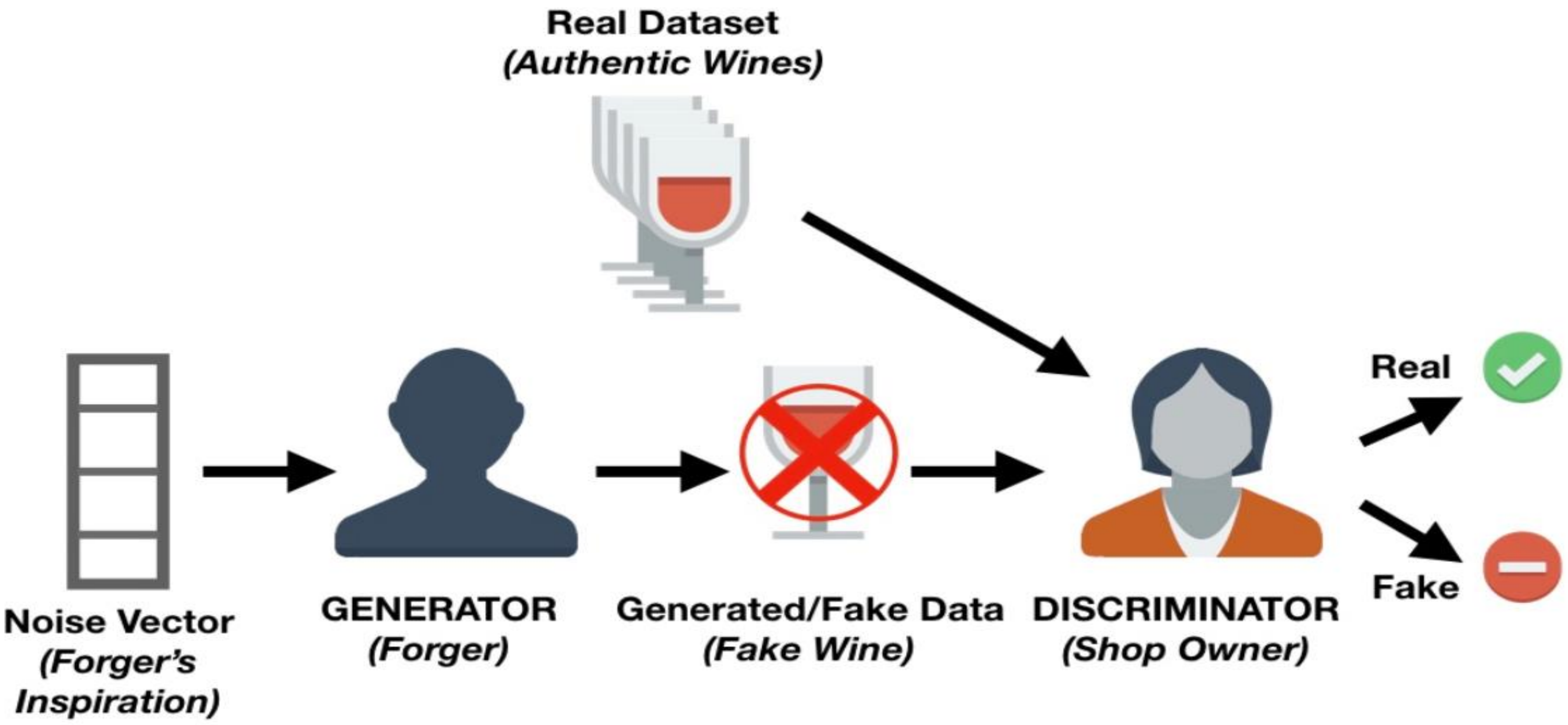


Image Source: https://www.datacamp.com/community/tutorials/generative-adversarial-networks

Working of GAN

Generative Adversarial Networks (GANs) can be broken down into three parts:

- **Generative:** To learn a generative model, which describes how data is generated in terms of a probabilistic model.
- **Adversarial:** The training of a model is done in an adversarial setting.
- **Networks:** Use deep neural networks as algorithms for training purpose.

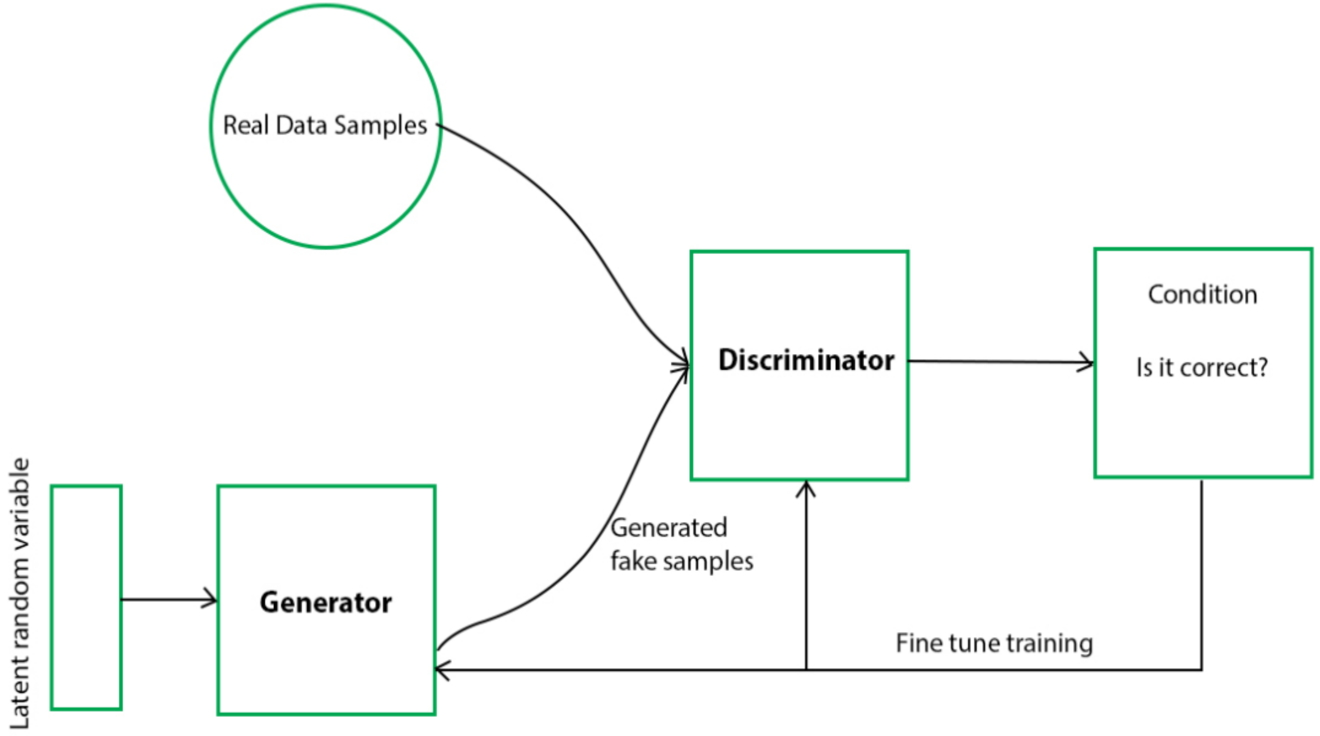


Image Source: https://www.geeksforgeeks.org/generative-adversarial-network-gan/

- ❑ The Generator generates fake samples of data and tries to fool the Discriminator.
- ❑ The Discriminator, on the other hand, tries to distinguish between the real and fake samples. The Generator and the Discriminator are both Neural Networks and they both run in competition with each other in the training phase. The steps are repeated several times
- ❑ The Generator and Discriminator get better and better in their respective jobs after each repetition.

Working of GAN

❖ The GANs are expressed as a minimax game, where the Discriminator is trying to minimize its reward **V(D, G)** and the Generator is trying to minimize the Discriminator's reward or in other words, maximize its loss. It can be mathematically described by the formula below:

$$\min_{G} \max_{D} V(D, G)$$

$$V(D, G) = \mathbb{E}_{x \sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$$

where,
G = Generator
D = Discriminator
Pdata(x) = distribution of real data
P(z) = distribution of generator
x = sample from Pdata(x)
z = sample from P(z)
D(x) = Discriminator network
G(z) = Generator network

Image Source: https://www.geeksforgeeks.org/generative-adversarial-network-gan/

Working of GAN

Training of a GAN has two parts:

- **Part 1:** The Discriminator is trained while the Generator is idle.

  In this phase, the network is only forward propagated and no back-propagation is done.

  The Discriminator is trained on real data for n epochs, and see if it can correctly predict them as real. The Discriminator is also trained on the fake generated data from the Generator and see if it can correctly predict them as fake.
- **Part 2:** The Generator is trained while the Discriminator is idle.

  After the Discriminator is trained by the generated fake data of the Generator, we can get its estimates and use the results for training the Generator and get better from the previous state to try and fool the Discriminator.

  The above method is repeated for a few epochs and then manually check the fake data if it seems genuine.

  If it seems acceptable, then the training is stopped, otherwise, its allowed to continue for few more epochs.

Different types of GANs:

- ✓ Vanilla GAN
- ✓ Conditional GAN (CGAN)
- ✓ Deep Convolutional GAN (DCGAN)
- ✓ Laplacian Pyramid GAN (LAPGAN)
- ✓ Super Resolution GAN (SRGAN)

Image Source: https://www.geeksforgeeks.org/generative-adversarial-network-gan/

Convolution Neural Networks (CNN)

- ❑ A **Convolutional Neural Network (ConvNet/CNN)** is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other.
- ❑ The pre-processing required in a ConvNet is much lower as compared to other classification algorithms.
- ❑ In other methods filters are hand-engineered, with enough training, but ConvNets have the ability to learn these filters/characteristics.

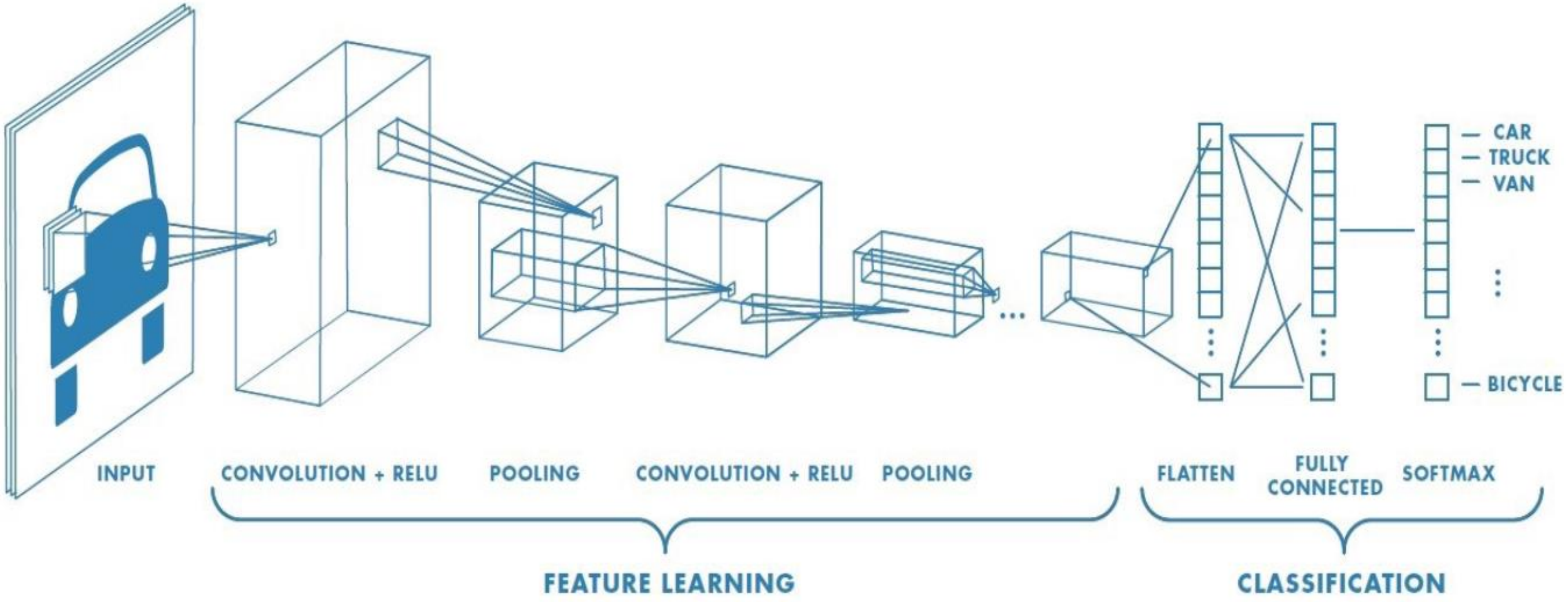
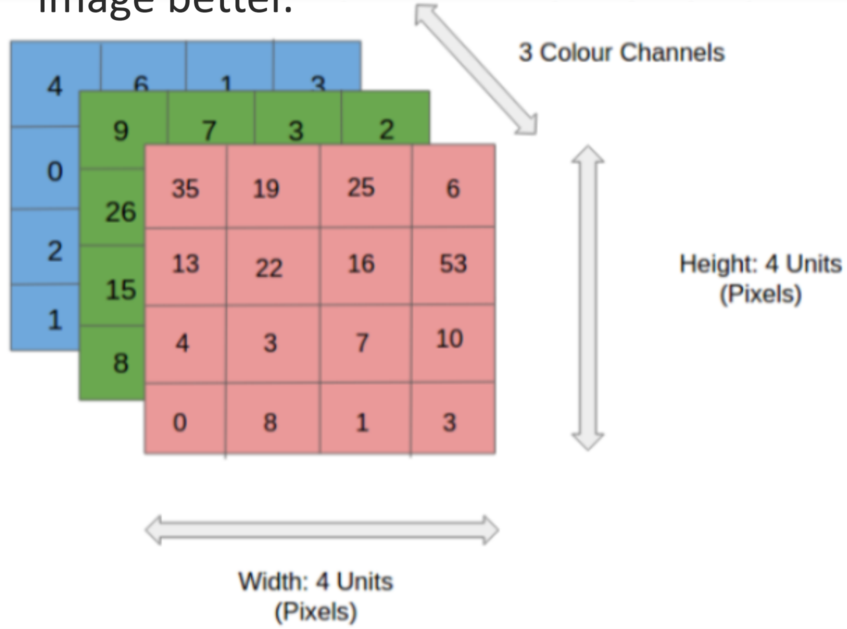


Image Source: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Convolution Neural Networks (CNN)

- ❑ The architecture of a ConvNet is analogous to that of the connection pattern of Neurons present in Human Brain and was inspired by the organization of the Visual Cortex region.
- ❑ Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field.
- ❑ A collection of such fields overlap to cover the entire visual area.
- ❑ A ConvNet is able to **successfully capture the Spatial and Temporal dependencies** in an image through the application of relevant filters.
- ❑ The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better.



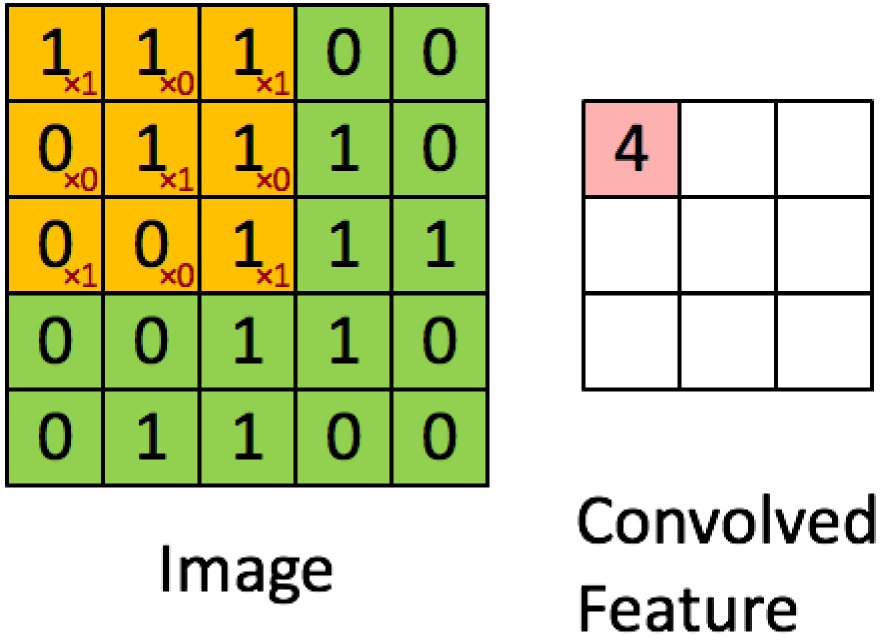
❖ In the figure, we have an RGB image which has been separated by its three color planes — Red, Green, and Blue

➢ The role of the ConvNet is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction

Image Source: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Convolution Neural Networks (CNN)

❖ Convolution Layer:



$1_{\times1}$	$1_{\times0}$	$1_{\times1}$	0	0
$0_{\times0}$	$1_{\times1}$	$1_{\times0}$	1	0
$0_{\times1}$	$0_{\times0}$	$1_{\times1}$	1	1
0	0	1	1	0
0	1	1	0	0

Image

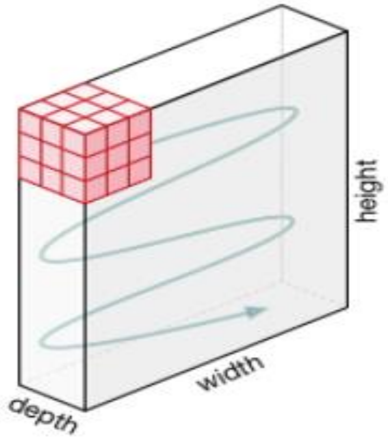
4		

Convolved Feature

1	0	1
0	1	0
1	0	1

Kernel / FIlter

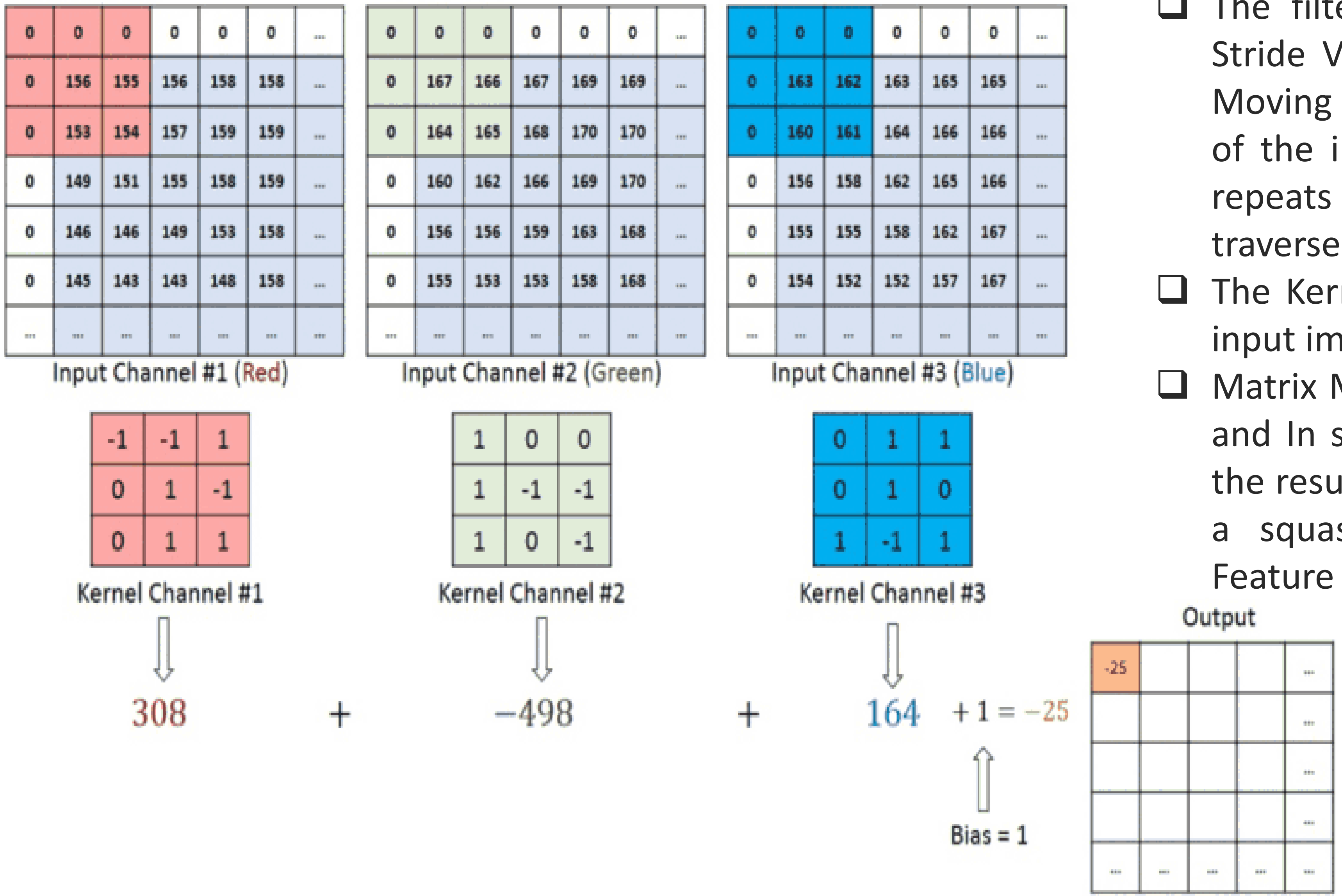
- Consider an Image with Dimensions = 5 (Height) x 5 (Breadth) x 1 (Number of channels, eg. RGB)
- The green section denotes our **5x5x1 input image, I**.
- The element involved in carrying out the convolution operation in the first part of a Convolutional Layer is called the **Kernel/Filter, K**, represented in the color yellow. We have selected **K as a 3x3x1 matrix.**
- The Kernel shifts 9 times because of **Stride Length = 1 (Non-Strided)**, every time performing a **matrix multiplication operation between K and the portion P of the image** over which the kernel is hovering.



Movement of the Kernel

Image Source: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Convolution Neural Networks (CNN)



0	0	0	0	0	0	...
0	156	155	156	158	158	...
0	153	154	157	159	159	...
0	149	151	155	158	159	...
0	146	146	149	153	158	...
0	145	143	143	148	158	...
...	...	...	...	...	...	...

Input Channel #1 (Red)

0	0	0	0	0	0	...
0	167	166	167	169	169	...
0	164	165	168	170	170	...
0	160	162	166	169	170	...
0	156	156	159	163	168	...
0	155	153	153	158	168	...
...	...	...	...	...	...	...

Input Channel #2 (Green)

0	0	0	0	0	0	...
0	163	162	163	165	165	...
0	160	161	164	166	166	...
0	156	158	162	165	166	...
0	155	155	158	162	167	...
0	154	152	152	157	167	...
...	...	...	...	...	...	...

Input Channel #3 (Blue)

-1	-1	1
0	1	-1
0	1	1

Kernel Channel #1

1	0	0
1	-1	-1
1	0	-1

Kernel Channel #2

0	1	1
0	1	0
1	-1	1

Kernel Channel #3

$$308 + -498 + 164 + 1 = -25$$

Bias = 1

Output

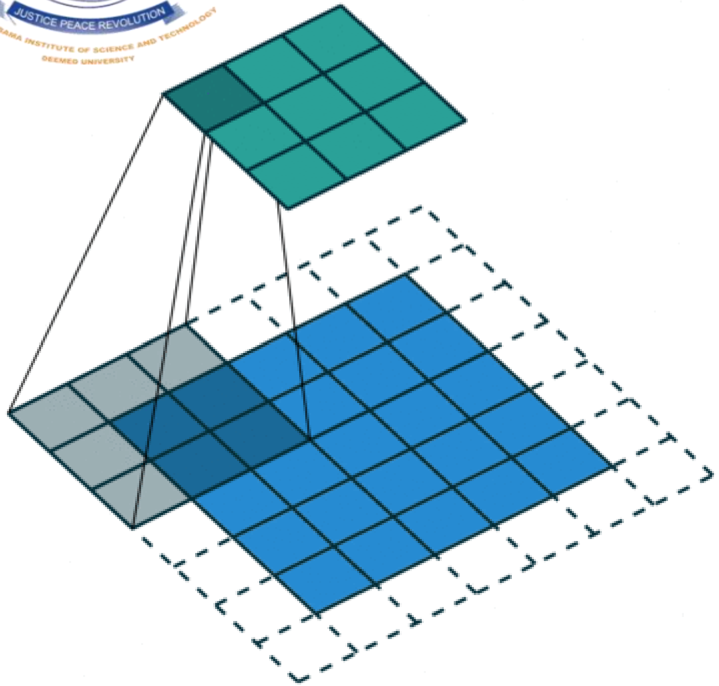
-25				...
				...
				...
				...
...	...	...	...	...

- ❑ The filter moves to the right with a certain Stride Value till it parses the complete width. Moving on, it hops down to the beginning (left) of the image with the same Stride Value and repeats the process until the entire image is traversed.
- ❑ The Kernel has the same depth as that of the input image.
- ❑ Matrix Multiplication is performed between Kn and In stack ([K1, I1]; [K2, I2]; [K3, I3]) and all the results are summed with the bias to give us a squashed one-depth channel Convoluted Feature Output.

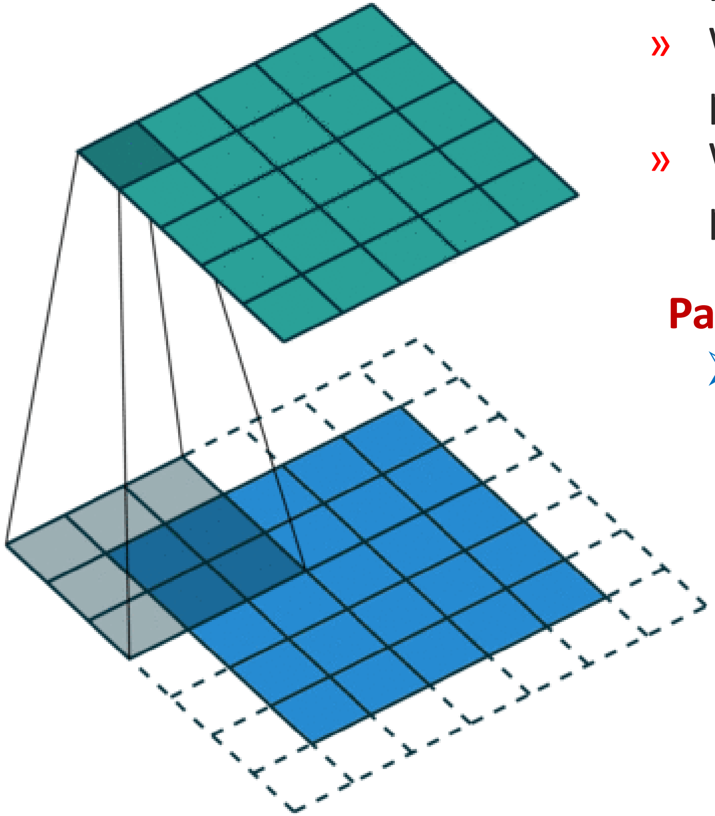
➢ The objective of the Convolution Operation is to **extract the high-level features** such as edges, from the input image

Image Source: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Convolution Neural Networks (CNN)



Convolution Operation with Stride Length = 2



Convolution Operation with Stride Length = 1

- Stride is the number of pixels shifts over the input matrix.
- When the stride is 1 then we move the filters to 1 pixel at a time.
- When the stride is 2 then we move the filters to 2 pixels at a time and so on

Padding :

- If the filter does not fit perfectly fit the input image. We have two options:
  1. Pad the picture with zeros (zero-padding) so that it fits
  2. Drop the part of the image where the filter did not fit. This is called valid padding which keeps only valid part of the image.

Image Source: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

CNN - Pooling Layer

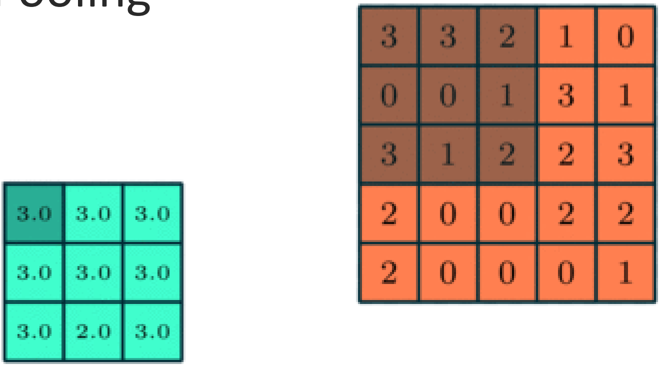
» **Pooling layers** section would reduce the number of parameters when the images are too large.

» Spatial pooling also called subsampling or down sampling which reduces the dimensionality of each map but retains important information

» This is to **decrease the computational power required to process the data** by reducing the dimensions

Types of Pooling:

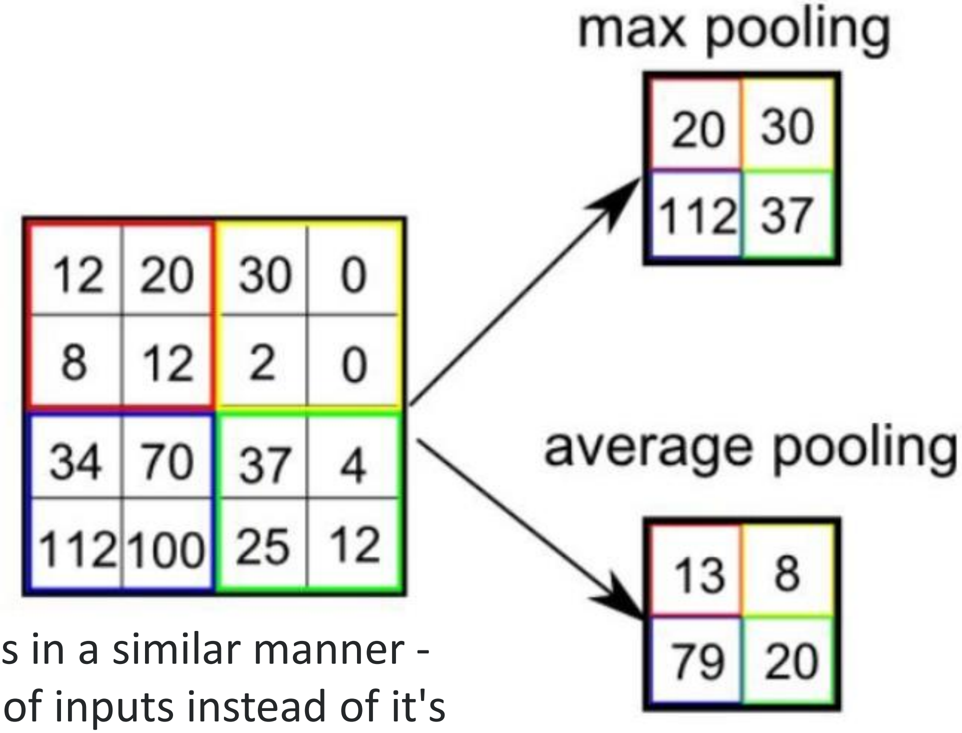
- Max Pooling
- Average Pooling
- Sum Pooling



3.0	3.0	3.0
3.0	3.0	3.0
3.0	2.0	3.0

3	3	2	1	0
0	0	1	3	1
3	1	2	2	3
2	0	0	2	2
2	0	0	0	1

Max Pooling



12	20	30	0
8	12	2	0
34	70	37	4
112	100	25	12

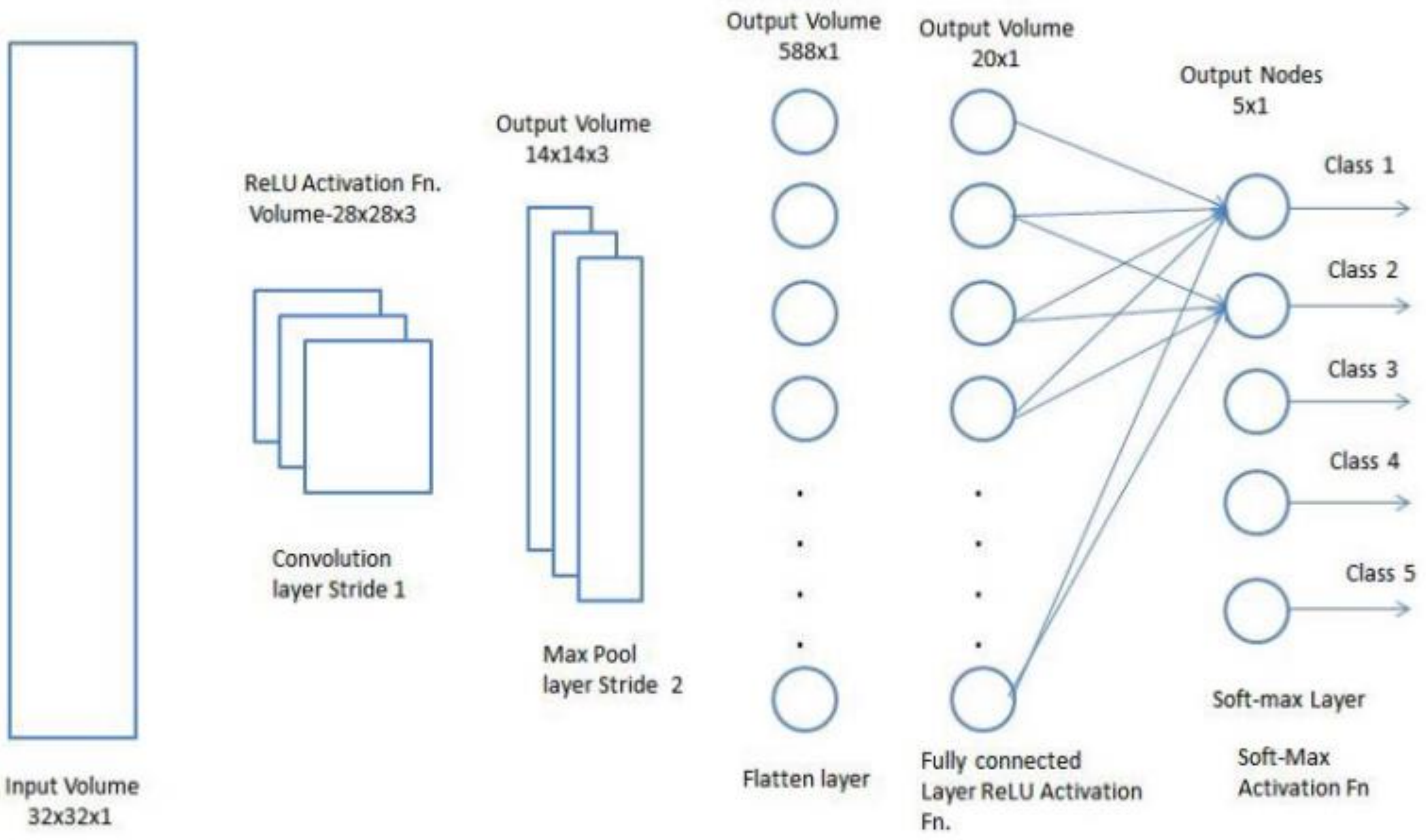
20	30
112	37

13	8
79	20

Sum pooling works in a similar manner - by taking the sum of inputs instead of it's maximum.

Image Source: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

CNN - Classification — Fully Connected Layer (FC Layer)



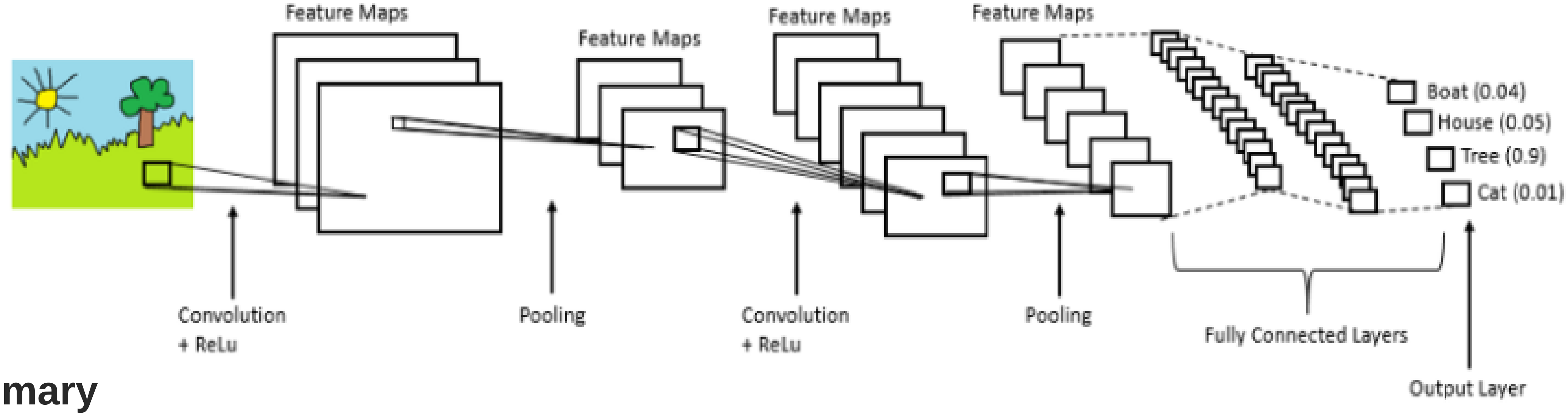
Max Pooling

- ❑ The image is flattened into a column vector.
- ❑ The flattened output is fed to a feed-forward neural network and backpropagation applied to every iteration of training.
- ❑ Over a series of epochs, the model is able to distinguish between dominating and certain low-level features in images and classify them using the **Softmax Classification** technique.

The feature map matrix will be converted as vector (x1, x2, x3, …).
These features are combined together to create a model.
Finally, an activation function such as softmax or sigmoid is used to classify the outputs as cat, dog, car, truck etc.

Image Source: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

CNN - Summary



Summary

- Provide input image into convolution layer
- Choose parameters, apply filters with strides, padding if requires. Perform convolution on the image and apply ReLU activation to the matrix.
- Perform pooling to reduce dimensionality size
- Add as many convolutional layers until satisfied
- Flatten the output and feed into a fully connected layer (FC Layer)
- Output the class using an activation function (Logistic Regression with cost functions) and classifies images.

Image Source: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148

Dr. V. Vedanarayanan B.E., M.E., PhD
Assistant Professor, Department of ECE,
School of Electrical and Electronics
Sathyabama Institute of Science and technology
Vedanarayanan.etc@sathyabama.ac.in