Linear Regression

Objectives

- What is machine learning
- Types of data and terminology
- Types of machine learning
- Supervised learning
- Linear regression
- Least square and Gradient Descent
- Hands on implementing Linear Regression from sketch.

Machine Learning

- Machine Learning is the science to make computers learn from data without explicitly program them and improve their learning over time in autonomous fashion.

- This learning comes by feeding them **data** in the form of observations and real-world interactions.”

- Machine Learning can also be defined as a tool to **predict** future events or values using past **data**.

Types of Data

- ***Based on Values***
  - *Continuous data (ex. Age – 0-100)*
  - *Categorical data (ex. Gender- Male/Female)*
- ***Based on pattern***
  - *Structured data (ex. Databases)*
  - *Unstructured data (ex. Audio, Video, Text)*

Types of Data- continued

- ***Labelled data*** *– consists of input output pair. For every set input features the output/response/label is present in dataset. (ex- labelled image as cat's or dog's photo)*

$$\{(x_1, y_1), (x_2, y_2), (x_3, y_3) \dots \dots \dots \dots \dots (x_n, y_n)\}$$

- ***Unlabelled data****- There is no output/response/label for the input features in data. (ex. news articles, tweets, audio)*

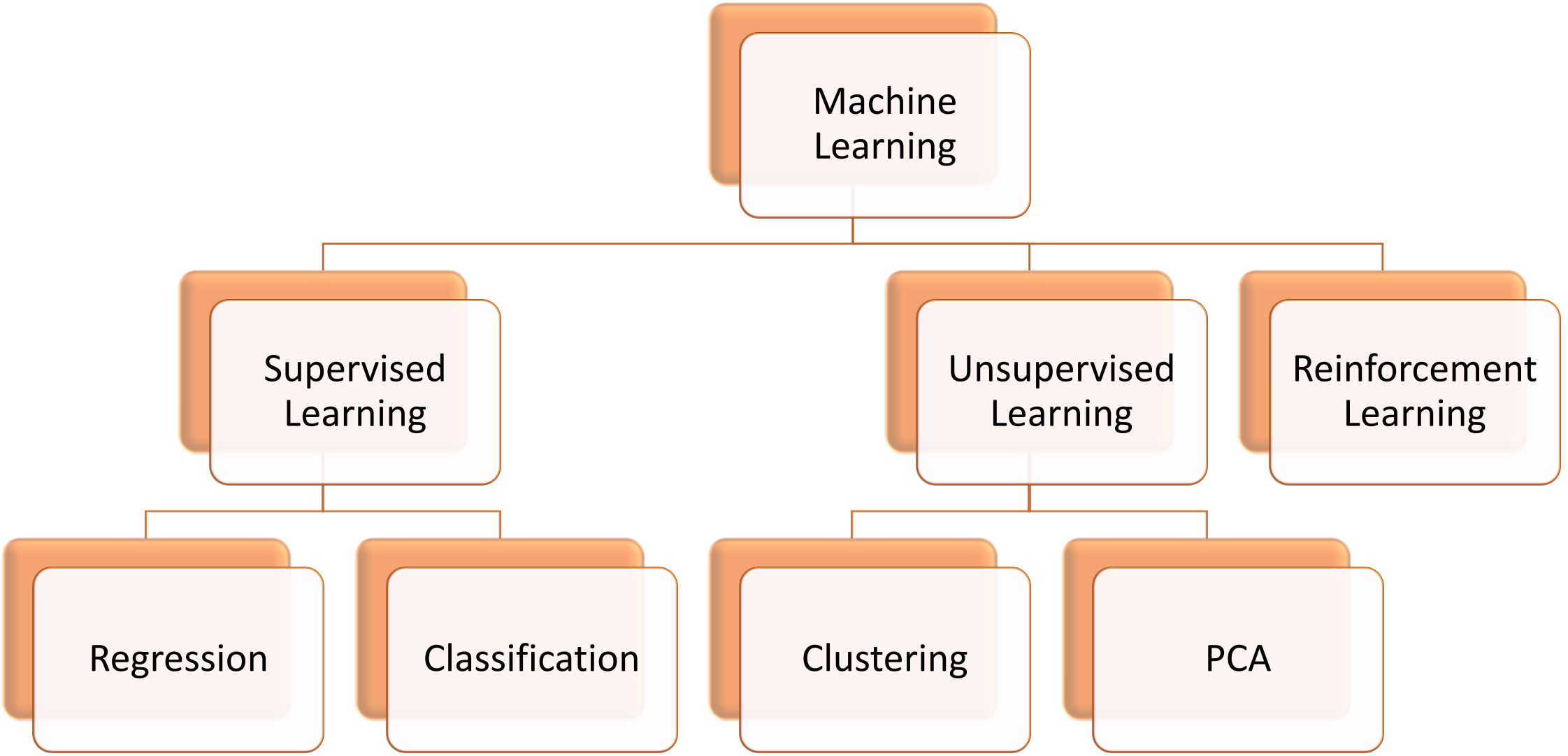
$$\{x_1, x_2, x_3 \dots \dots \dots \dots x_n\}$$

Types of Data- continued

- ***Training Data*** *– Sample data points which are used to train the machine learning model.*
- ***Test Data****- sample data points that are used to test the performance of machine learning model.*

Note- For modelling, the original dataset is partitioned into the ratio of 70:30 or 75:25 as training data and test data.

Types of Machine Learning

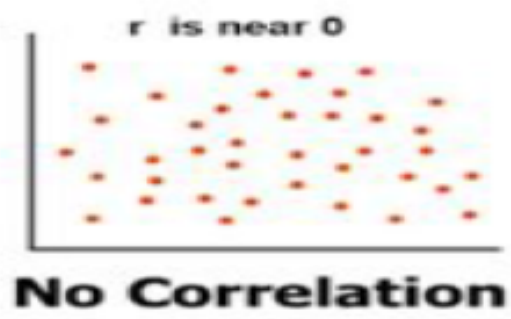


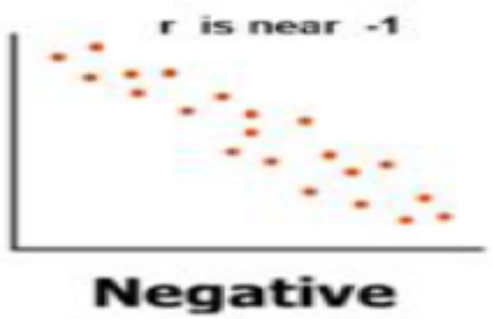
Supervised Learning

- Class of machine learning that work on externally supplied instances in form of predictor attributes and **associated target values**.
- The model learns from the training data using these '**target variables'** as reference variables.
  - Ex1 : *model to predict the resale value of a car based on its mileage, age, color etc.*
- The **target values** are the 'correct answers' for the predictor model which can either be a **regression model** or a **classification model.**

Motivation for learning

- It is being assumed that there exists a relationship/association between **input features** and **target variable.**
- Relationship can be observed by plotting a scatter plot between the two variables.



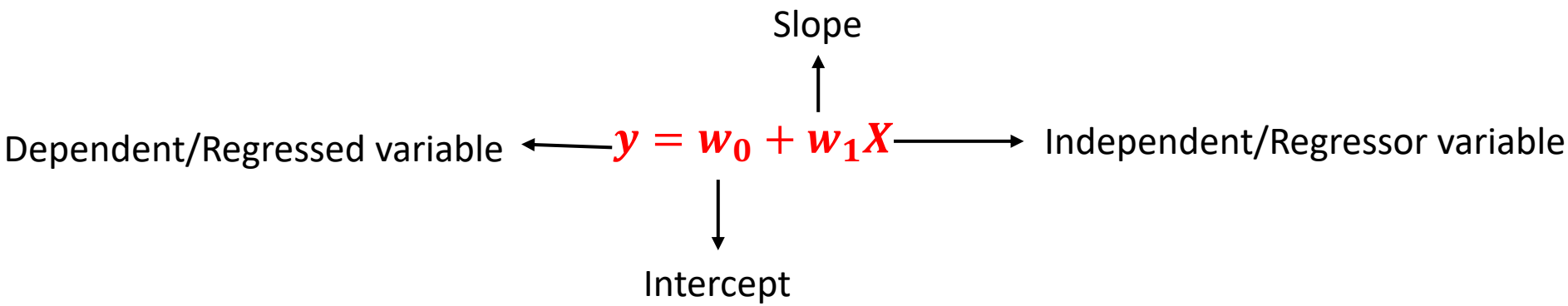


- Relationship measure can be quantified by calculating correlation between two the variables.

$$corr(x,y) = \frac{cov(x,y)}{var(x).var(y)} = \frac{\sum(x_i - \bar{x}) * (y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2}\sqrt{\sum(y_i - \bar{y})^2}}$$

Linear Regression

- Linear regression is a way to identify a relationship between two or more variables and use these relationships to predict values of one variable for given value(s) of other variable(s).
- Linear regression assume the relationship between variables can be modelled through linear equation or an equation of line



Multiple Regression

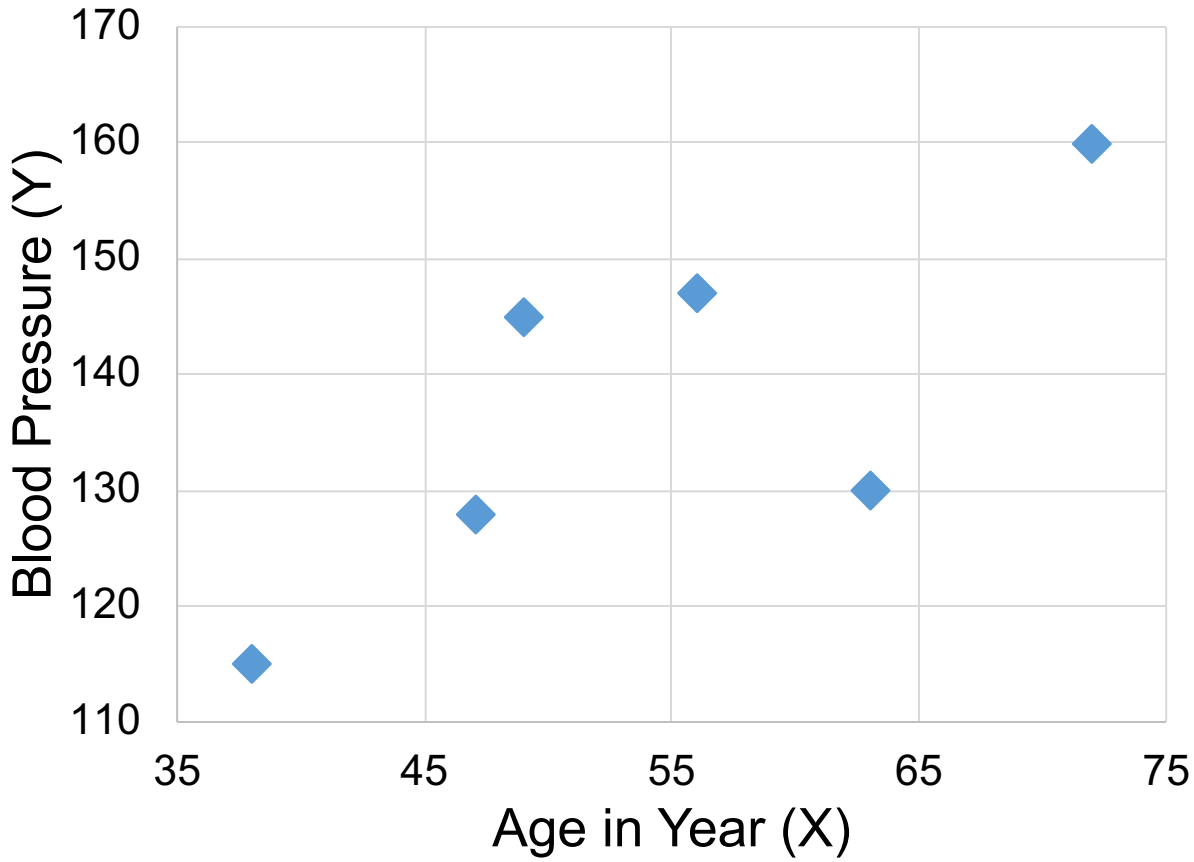
- Last slide showed the linear regression model with one independent and one dependent variable.
- In Real world a data point has various important attributes and they need to be catered to while developing a regression model. (Many independent variables and one dependent variable)

$$y = w_0 + w_1x_1 + w_2x_2 + w_3x_3 \ldots\ldots\ldots\ldots \mathrm{w_n x_n}$$

Regression –Problem Formulation

Let you have given with a data:

Age in Years **(X)**	Blood Pressure **(Y)**
56	147
49	145
72	160
38	115
63	130
47	128



Linear Regression

❖ For given example the Linear Regression is modeled as:

$$BloodPressure(y) = w_0 + w_1 AgeinYear(X)$$

OR

$$y = w_0 + w_1 X$$ – Equation of line

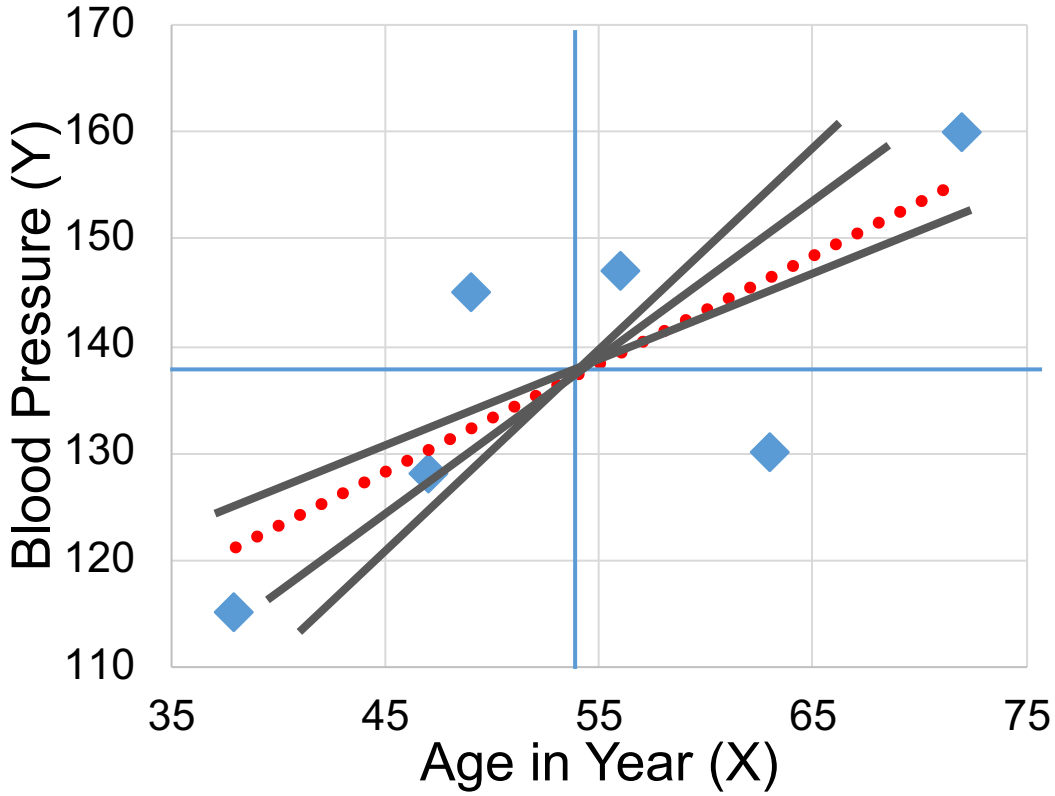
with w_0 $is\ intercept\ on\ Y\_axis\ and\ w_1$ $is\ slope\ of\ line$

Blood Pressure - **Dependent Variable**

Age in Year - **Independent Variable**

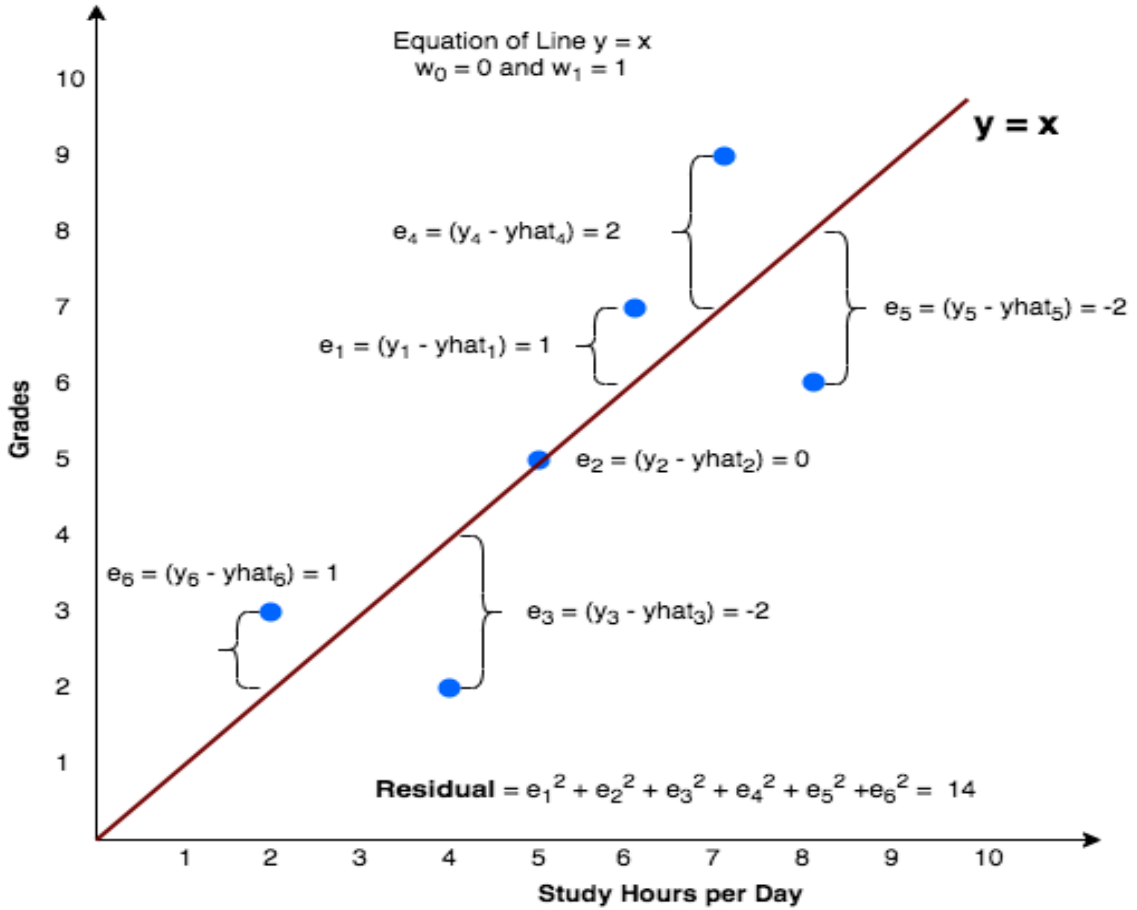
Linear Regression- Best Fit Line

- Regression uses line to show the trend of distribution.
- There can be many lines that try to fit the data points in scatter diagram
- The aim is to find **Best fit** Line



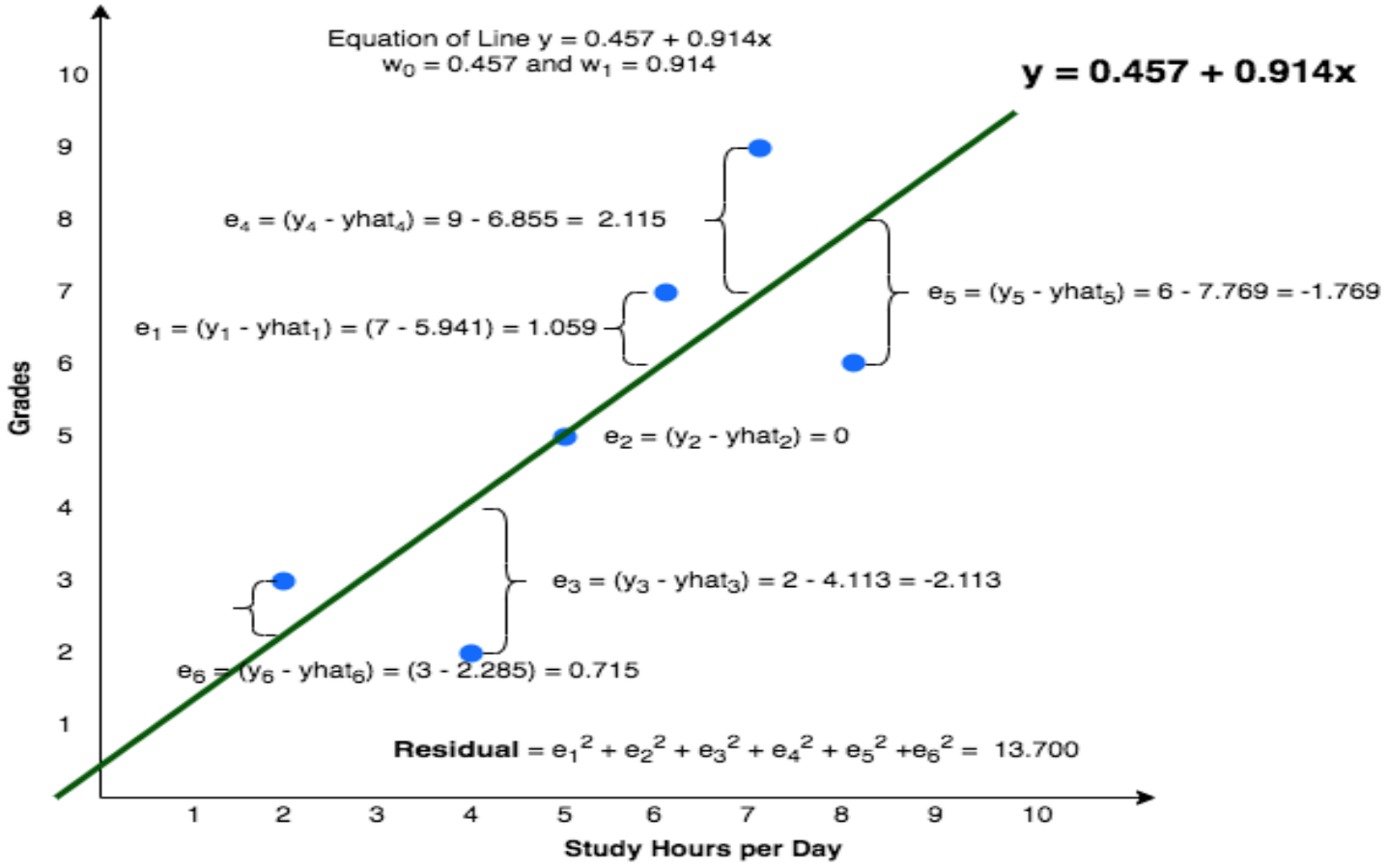
What is Best Fit Line

- Best fit line tries to explain the variance in given data. (minimize the total residual/error)



What is Best Fit Line

- Best fit line tries to explain the variance in given data. (minimize the total residual/error)



Linear Regression- Methods to Get Best

- Least Square
- Gradient Descent

Linear Regression- Least Square

Model: $Y = w_0 + w_1 X$

Task: Estimate $the\ value\ of\ w_0\ and\ w_1$

According to pr$inciple\ of\ least\ square\ the\ normal\ equations\ to\ solve\ for\ w_0\ and\ w_1$

$$\sum_{i=1}^{n} Y_i = n\, w_0 + w_1 \sum_{i=1}^{n} X_i \ldots\ldots\ldots\ldots\ldots (1)$$

$$\sum_{i=1}^{n} X_i Y_i = w_0 \sum_{i=1}^{n} X_i + w_1 \sum_{i=1}^{n} X_i^2 \ldots\ldots\ldots\ldots . (2)$$

Linear Regression–Least Square

Let divide the equation (1) by *n* (number of sample points) we get:

$$\frac{1}{n}\sum_{i=1}^{n} Y_i = w_0 + w_1 \frac{1}{n}\sum_{i=1}^{n} X_i$$

OR

$$\bar{y} = w_0 + w_1 \bar{x} \ldots\ldots\ldots\ldots (3)$$

So line of regression will always passes through the points $(\bar{x}, \bar{y})$

Linear Regression–Least Square

Now we know :

$$cov(x,y) = \frac{1}{n}\sum_{i=1}^{n} x_i y_i - \bar{x}\bar{y} := \frac{1}{n}\sum_{i=1}^{n} x_i y_i = cov(x,y) + \bar{x}\bar{y} \ldots\ldots\ldots(4)$$

and

$$var(x) = \frac{1}{n}\sum_{i=1}^{n} x_i^2 - \bar{x}^2 \quad \text{and} \quad var(y) = \frac{1}{n}\sum_{i=1}^{n} y_i^2 - \bar{y}^2$$

Dividing equation (2) by n and using equation (4) and (5) we get:

$$cov(x,y) + \bar{x}\bar{y} = w_0\bar{x} + w_1(var(x) + \bar{x}^2) \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(5)$$

Linear Regression–Least Square

Now by using equation

$$\bar{y} = w_0 + w_1\bar{x}$$

and

$$cov(x, y) + \bar{x}\bar{y} = w_0\bar{x} + w_1(var(x) + \bar{x}^2)$$

We will get:

$$w_1 = \frac{cov(x, y)}{var(x)}$$

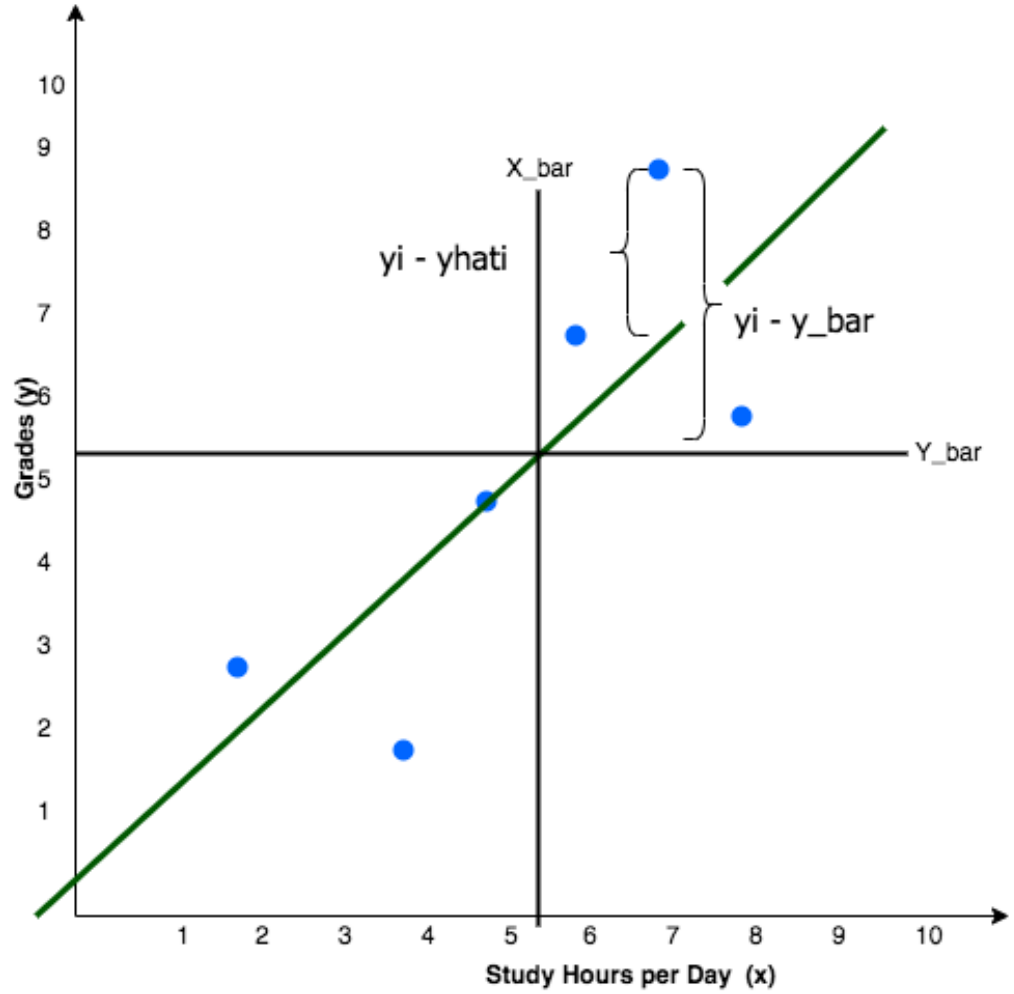
and

$$w_0 = \bar{y} - w_1\bar{x}$$

Performance metric for least square regression

$$R^2 = 1 - \frac{\frac{1}{n}\sum_{i=1}^{n}(y_i - yhat_i)^2}{\frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

$$R^2_{adj} = 1 - \frac{(1 - R^2)(n-1)}{(n - k - 1)}$$



Linear Regression- Gradient Descent

Model: $Y = w_0 + w_1 X$

Task: Estimate $the\ value\ of\ w_0\ and\ w_1$

Define the cost function,

$$cost(w_0, w_1) = \frac{1}{n}\sum_{i=1}^{n}(y_i - yhat_i)^2$$

Objective of gradient Descent

$$\min_{w_0, w_1} cost(w_0, w_1) = \frac{1}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i))^2$$

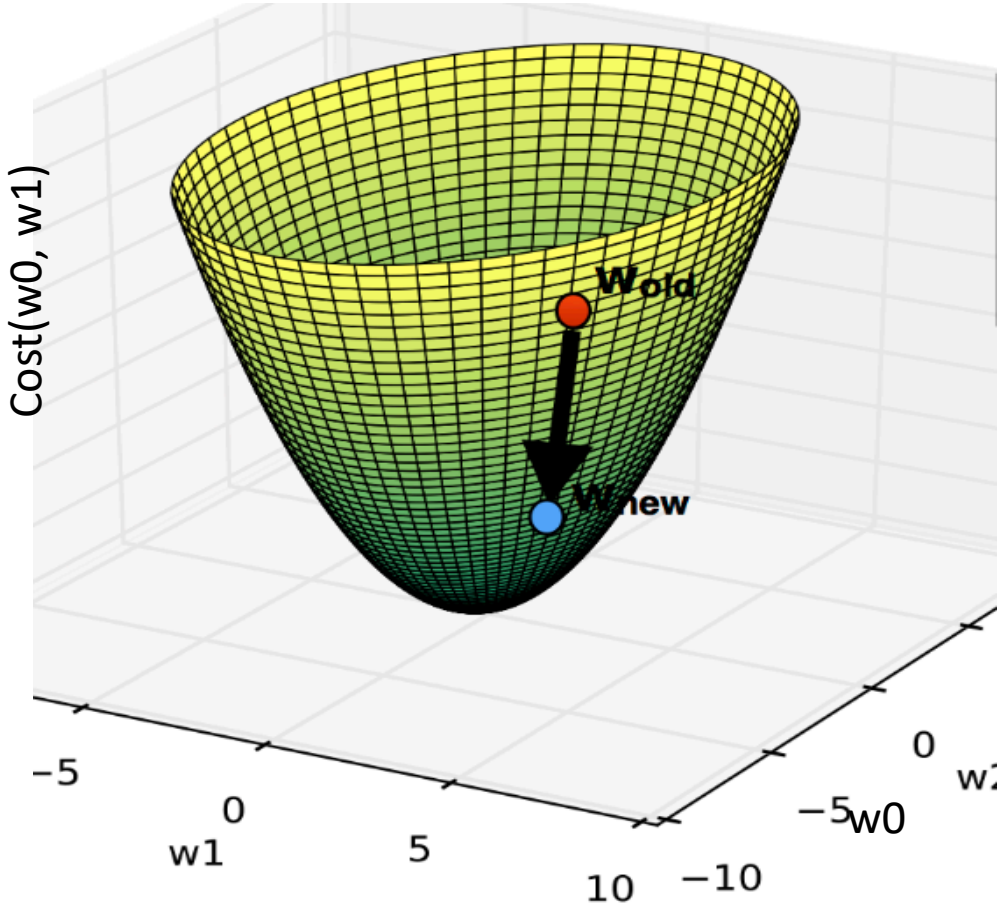
Linear Regression- Gradient Descent

Model: $Y = w_0 + w_1 X$

Task: Estimate $the\ value\ of\ w_0\ and\ w_1$

the objective,

$$\min_{w_0, w_1} cost(w_0, w_1) = \frac{1}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i))^2$$



Linear Regression- Gradient Descent

- Gradient descent works if following steps:

1. Initialize the parameters to some random variable
2. Calculate the gradient of cost function w. r. t. to parameters
3. Update the parameters using gradient in opposite direction.
4. Repeat step-2 and step-3 for some number of times or till it reaches to minimum cost value.

Linear Regression- Gradient Descent

$$cost(w_0, w_1) = \frac{1}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i))^2$$

Calculating gradients of cost function:

$$gradw_0 = \frac{\partial cost(w_0, w_1)}{\partial w_0} = \frac{2}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i))(-1)$$

$$gradw_1 = \frac{\partial cost(w_0, w_1)}{\partial w_1} = \frac{2}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i))(-x)$$

Parameter update:

$$w_0 = w_0 - learningrate * gradw_0$$

$$w_1 = w_1 - learningrate * gradw_1$$

Performance metric for gradient based regression

Root Mean Square Error (**RMSE**) is the standard deviation of prediction errors.

$$RMSE = \sqrt{\frac{(y_i - yhat_i)^2}{n}}$$

Mean absolute error (MAE) is a measure of difference between two variables.

$$MAE = \frac{|y_i - yhat_i|}{n}$$

Thank you !

Let see the hands on…

Regression

(Parametric Data Reduction: Regression and Log-Linear Models)

Linear regression

- Data modeled to fit a straight line
- Often uses the least-square method to fit the line

Multiple regression

- Allows a response variable Y to be modeled as a linear function of multidimensional feature vector

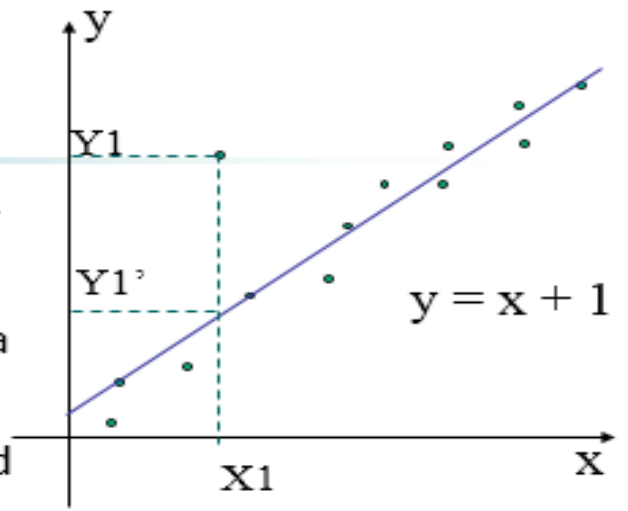
Log-linear model

- Approximates discrete multidimensional probability distributions

Regression Analysis

Regression Analysis

- Regression analysis: A collective name for techniques for the modeling and analysis of numerical data consisting of values of a ***dependent variable*** (also called ***response variable*** or *measurement*) and of one or more *independent variables* (aka. ***explanatory variables*** or ***predictors***)
- The parameters are estimated so as to give a "**best fit**" of the data
- Most commonly the best fit is evaluated by using the ***least squares method***, but other criteria have also been used



- Used for prediction (including forecasting of time-series data), inference, hypothesis testing, and modeling of causal relationships

Regress Analysis and Log-Linear Models

- Linear regression: $Y = w X + b$
  - Two regression coefficients, *w* and *b,* specify the line and are to be estimated by using the data at hand
  - Using the least squares criterion to the known values of $Y_1, Y_2, \ldots, X_1, X_2, \ldots$.
- Multiple regression: $Y = b_0 + b_1 X_1 + b_2 X_2$
  - Many nonlinear functions can be transformed into the above
- Log-linear models:
  - The technique is used for both hypothesis testing and model building.
  - Log-linear analysis is a technique used in statistics to examine the relationship between more than two categorical variables.
  - Log-linear analysis is a multidimensional extension of the classical cross-tabulation chi-square test .
  - Approximate discrete multidimensional probability distributions
  - Estimate the probability of each point (tuple) in a multi-dimensional space for a set of discretized attributes, based on a smaller subset of dimensional combinations

Linear Algebra

- Linear algebra is the branch of mathematics concerning vector spaces and linear mappings between such spaces.

➢ Why we do study linear algebra?

- Provides a way to compactly represent and operate on sets of linear equations.
- In machine learning, we represents data as a matrices.

➢ Consider the following system of the equations:

4X1-5X2= -13

-2X1+3X2=9

In Matrix Notation ,

- Consider the following system of equations:

$$4x_1 - 5x_2 = -13$$
$$-2x_1 + 3x_2 = 9$$

- In matrix notation, the system is more compactly represented as:

$$Ax = b$$

$$A = \begin{bmatrix} 4 & -5 \\ -2 & 3 \end{bmatrix}$$

$$b = \begin{bmatrix} -13 \\ 9 \end{bmatrix}$$

Applications of Linear Algebra

- Matrices in Engineering, such as a line of springs.
- Graphs and Networks, such as analyzing networks.
- **Markov Matrices**, Population, and Economics, such as population growth.
- Linear Programming, the simplex **optimization** method.
- **Fourier Series:** Linear Algebra for functions, used widely in signal processing.
- Linear Algebra for **statistics and probability**, such as least squares for regression.
- **Computer Graphics**, such as the various translation, rescaling and rotation of images.
- Application of linear algebra is that it is the type of mathematics used by **Albert Einstein** in parts of his theory of relativity.

Examples of Linear algebra in Lachine Learning

- Dataset and Data Files
- Images and Photographs
- One Hot Encoding
- Linear Regression
- Regularization
- Principal Component Analysis (PCA)
- Singular-Value Decomposition (SVD)
- Latent Semantic Analysis
- Recommender Systems

Mathematical Optimization

- Mathematical optimization is the selection of a best element from set of available alternatives.

Represented as: min $f_{0(x)}$, x is optimization variable, and f_0 is objective function.

$f_{0(x)}$ is the minimum possible values in the feasible region.

➢Example:

- **Data Fitting:**

Variable: Parameter of the model.

Constraint: Parameter Limits, Prior information

Objective: Measures of fit(E.g. Minimizing of error)

Solving Optimization Problem

- Optimization is the very tough problems to solve.
- Optimization problems are classified in to various classes based on the properties of objectives and constraints.
- Some of these classes can be solved efficiently :

1)Linear program

II) Least Square problem

III) Convex Optimization Problem

Convex Optimization Problem

Convex Function

Definition

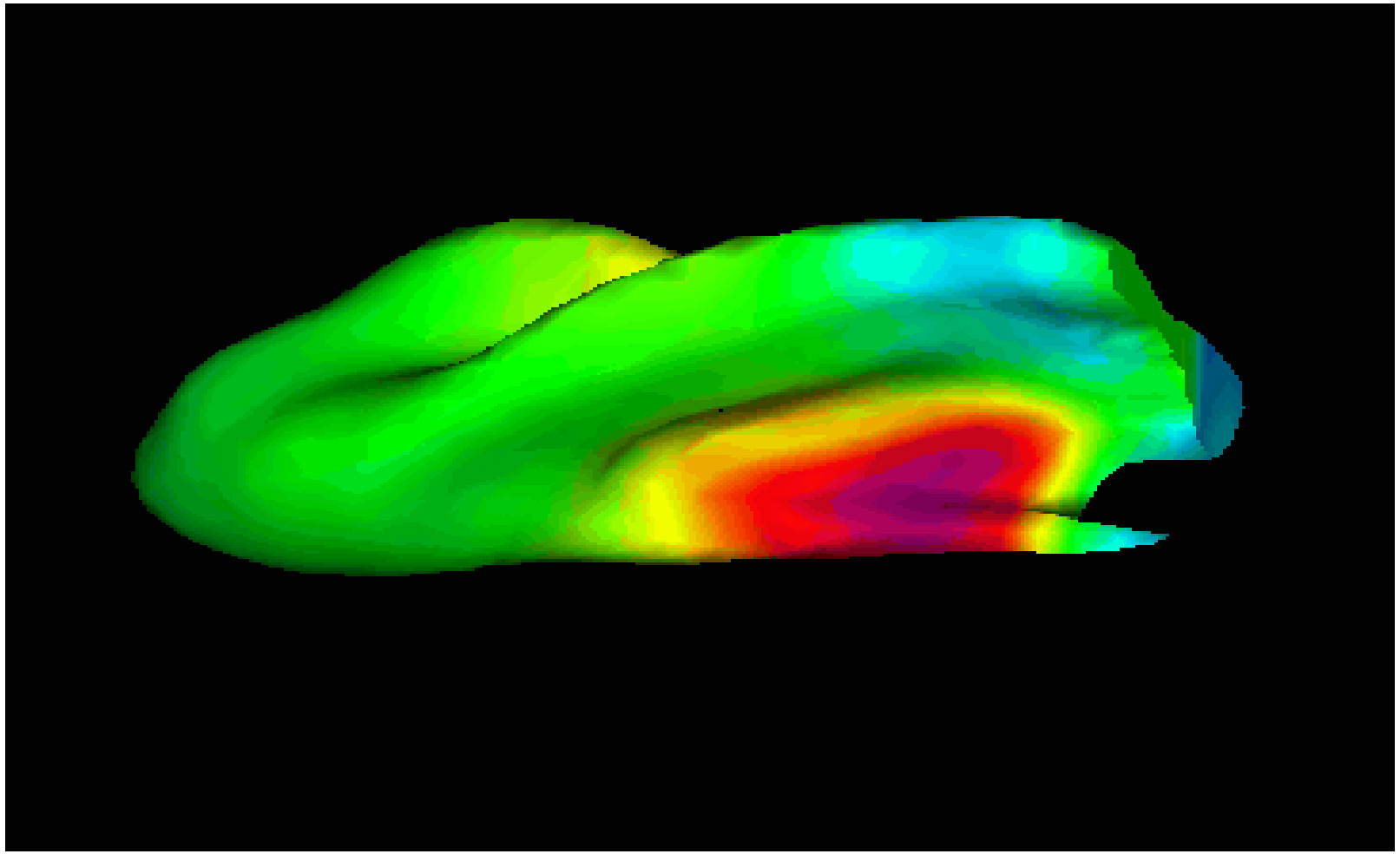
A function $f : \mathbb{R}^n \to \mathbb{R}$ is said to be convex if,

- Domain of f is a convex set
- $\forall x, y \in \text{dom}(f)$, and $0 \leq \theta \leq 1$, we have

$$f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta)f(y)$$

Data Visualization

- **“Seeing” the Data**



Visual Presentation

- For any kind of high dimensional data set, displaying predictive relationships is a challenge.
- The picture on the previous slide uses 3-D graphics to portray the weather balloon data numbers in text Table 11-4. We learn very little from just examining the numbers .
- Shading is used to represent relative degrees of thunderstorm activity, with the darkest regions the heaviest activity.

Market Basket Analysis

- This is the most widely used machine learning techniques.
- It essentially determines what products people purchase together.
- Stores can use this information to place these products in the same area.
- Direct marketers can use this information to determine which new products to offer to their current customers.
- Inventory policies can be improved if reorder points reflect the demand for the complementary products.

➢We first need a list of transactions and what was purchased. This is pretty easily obtained these days from scanning cash registers.

➢Next, we choose a list of products to analyze, and tabulate how many times each was purchased with the others.

➢The diagonals of the table shows how often a product is purchased in any combination, and the off-diagonals show which combinations were bought.

A Convenience Store Example (5 transactions)

Consider the following simple example about five transactions at a convenience store:

Transaction 1: Frozen pizza, cola, milk
Transaction 2: Milk, potato chips
Transaction 3: Cola, frozen pizza
Transaction 4: Milk, pretzels
Transaction 5: Cola, pretzels

These need to be cross tabulated and displayed in a table.

A Convenience Store Example (5 transactions)

A Convenience Store Example (5 transactions)

Product Bought	Pizza also	Milk also	Cola also	Chips also	Pretzels also
Pizza	2	1	2	0	0
Milk	1	3	1	1	1
Cola	2	1	3	0	1
Chips	0	1	0	1	0
Pretzels	0	1	1	0	2

- Pizza and Cola sell together more often than any other combo; a cross-marketing opportunity?
- Milk sells well with everything – people probably come here specifically to buy it.

Using the Results

- The tabulations can immediately be translated into association rules and the numerical measures computed.
- Comparing this week's table to last week's table can immediately show the effect of this week's promotional activities.
- Some rules are going to be *trivial* (hot dogs and buns sell together) or *inexplicable (*toilet rings sell only when a new hardware store is opened).
- **Limitations to Market Basket Analysis:**
- A large number of real transactions are needed to do an effective basket analysis, but the data's accuracy is compromised if all the products do not occur with similar frequency.
- The analysis can sometimes capture results that were due to the success of previous marketing campaigns (and not natural tendencies of customers).

Computing Measures of Association

Computing Measures of Association

	Pizza	Milk	Cola	Chips	Pretzels
Pizza	2	1	2	0	0
Milk	1	3	1	1	1
Cola	2	1	3	0	1
Chips	0	1	0	1	0
Pretzels	0	1	1	0	2

Let's do some of the textbook's example computations here ……

Data Preprocessing

- **Data quality**: accuracy, completeness, consistency, timeliness, believability, interpretability
- **Data cleaning**: e.g. missing/noisy values, outliers
- **Data integration** from multiple sources:
  - Entity identification problem
  - Remove redundancies
  - Detect inconsistencies
- **Data reduction**
  - Dimensionality reduction
  - Numerosity reduction
  - Data compression
- **Data transformation and data discretization**
  - Normalization
  - Concept hierarchy

Why Preprocess the Data?

- Measures for data quality: A multidimensional view
  - Accuracy: correct or wrong, accurate or not
  - Completeness: not recorded, unavailable, …
  - Consistency: some modified but some not, dangling, …
  - Timeliness: timely update?
  - Believability: how trustable the data are correct?
  - Interpretability: how easily the data can be understood?

Data Transformation

- A function that maps the entire set of values of a given attribute to a new set of replacement values s.t. each old value can be identified with one of the new values
- **Methods**
  - Smoothing: Remove noise from data
  - Attribute/feature construction
    - New attributes constructed from the given ones
  - **Aggregation:** Summarization, data cube construction
  - **Normalization:** Scaled to fall within a smaller, specified range
    - min-max normalization
    - z-score normalization
    - normalization by decimal scaling
  - **Discretization**: Concept hierarchy climbing

Normalizing data

Normalization

- **Min-max normalization**: to [new_min$_A$, new_max$_A$]

$$v' = \frac{v - min_A}{max_A - min_A}(new\_max_A - new\_min_A) + new\_min_A$$

  - Ex. Let income range \$12,000 to \$98,000 normalized to [0.0, 1.0]. Then \$73,000 is mapped to $\frac{73{,}600 - 12{,}000}{98{,}000 - 12{,}000}(1.0 - 0) + 0 = 0.716$

- **Z-score normalization** (μ: mean, σ: standard deviation):

$$v' = \frac{v - \mu_A}{\sigma_A}$$

  - Ex. Let μ = 54,000, σ = 16,000. Then $\frac{73{,}600 - 54{,}000}{16{,}000} = 1.225$

- **Normalization by decimal scaling**

$$v' = \frac{v}{10^j}$$ Where j is the smallest integer such that $\text{Max}(|v'|) < 1$

Discretization

- Three types of attributes
  - Nominal—values from an unordered set, e.g., color, profession
  - Ordinal—values from an ordered set, e.g., military or academic rank
  - Numeric—real numbers, e.g., integer or real numbers
- Discretization: Divide the range of a continuous attribute into intervals
  - Interval labels can then be used to replace actual data values
  - Reduce data size by discretization
  - Supervised vs. unsupervised
  - Split (top-down) vs. merge (bottom-up)
  - Discretization can be performed recursively on an attribute
  - Prepare for further analysis, e.g., classification

Data Augmentation

- **Data augmentation** is a **data-**depended process.
- In general, you need it when your **training data** is complex and you have a few samples.
- It can help, the **features** the network will learn to extract are easy and highly different from each other.
- It means increasing the number of data points.
- In terms of **images,** it may mean that increasing the number of images in the dataset.
- It adds value to base **data** by adding information derived from internal and external sources.
- **Crops random** regions of the source image according to a given range and speed, to generate a smooth animation.

Linearity vs non Linearity

Linearity:

Linearity and non linearity defined on the basis of activation function.

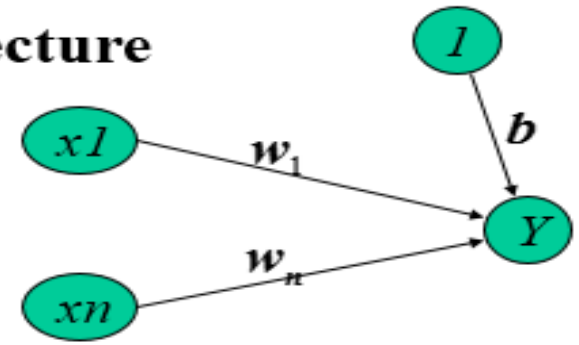
- In linear regression, data are modeled using linear predictor functions.
- linear methods involve only linear combinations of data, leading to easier implementations, etc.
- it is a statistical analytic technique used for separating sets of observed values and allocating new values.
- It is used PCA and creating an Index for missing data.
- **Non Linearity:** (why do we need Non-Linearities)
- Non-linear functions are those which have degree more than one and they have a curvature.
- when we plot a Non-Linear function. Now we need a Neural Network Model to learn and represent almost anything and any arbitrary complex function which maps inputs to outputs.
- Non linear techniques are ANN and SVM.
- They use, in general, non linear functions of the data (the activation function in ANN or the kernel in SVM .

General discussion

- **Pattern recognition**
  - Patterns: images, personal records, driving habits, etc.
  - Represented as a vector of features (encoded as integers or real numbers in NN)
  - Pattern classification:
    - Classify a pattern to one of the given classes
    - Form pattern classes
  - Pattern associative recall
    - Using a pattern to recall a related pattern
    - **Pattern completion**: using a partial pattern to recall the whole pattern
    - **Pattern recovery:** deals with noise, distortion, missing information

- **General architecture**

  Single layer



net input to Y: $net = b + \sum_{i=1}^{n} x_i w_i$

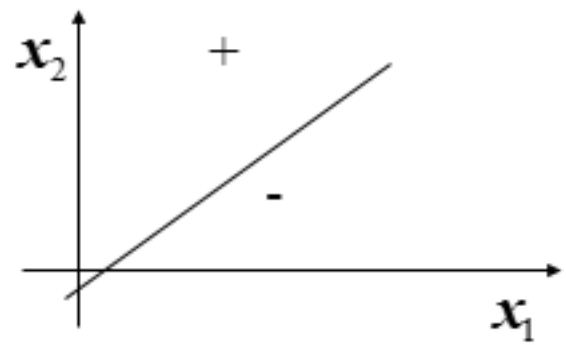
bias b is treated as the weight from a special unit with constant output 1.

threshold θ related to Y

output $y = f(net) = \begin{cases} 1 & \text{if } net \geq \theta \\ -1 & \text{if } net < \theta \end{cases}$

classify $(x_1, \ldots\ldots x_n)$ into one of the two classes

- **Decision region/boundary**



n = 2, b != 0, θ = 0

$b + x_1 w_1 + x_2 w_2 = 0$ or

$$x_2 = -\frac{w_1}{w_2} x_1 - \frac{b}{w_2}$$

is a line, called *decision boundary*, which partitions the plane into two decision regions

If a point/pattern (x_1, x_2) is in the positive region, then

$b + x_1 w_1 + x_2 w_2 \geq 0$, and the output is one (belongs to class one)

Otherwise, $b + x_1 w_1 + x_2 w_2 < 0$, output −1 (belongs to class two)

n = 2, b = 0, θ != 0 would result a similar partition

- If n = 3 (three input units), then the decision boundary is a two dimensional plane in a three dimensional space

- In general, a decision boundary $b + \sum_{i=1}^{n} x_i w_i = 0$ is a n-1

- dimensional hyper-plane in an n dimensional space, which partition the space into two decision regions

- This simple network thus can classify a given pattern into one of the two classes, provided one of these two classes is entirely in one decision region (one side of the decision boundary) and the other class is in another region.

- The decision boundary is determined completely by the weights $\boldsymbol{W}$ and the bias $\boldsymbol{b}$ (or threshold $\boldsymbol{q}$).

Linear Separability Problem

- If two classes of patterns can be separated by a decision boundary, represented by the linear equation $b + \sum_{i=1}^{n} x_i w_i = 0$

  then they are said to be linearly separable. The simple network can correctly classify any patterns.

- Decision boundary (i.e., ***W, b*** or θ) of linearly separable classes can be determined either by some learning procedures or by solving linear equation systems based on representative patterns of each classes.

- If such a decision boundary does not exist, then the two classes are said to be linearly inseparable.
- Linearly inseparable problems cannot be solved by the simple network , more sophisticated architecture is needed.

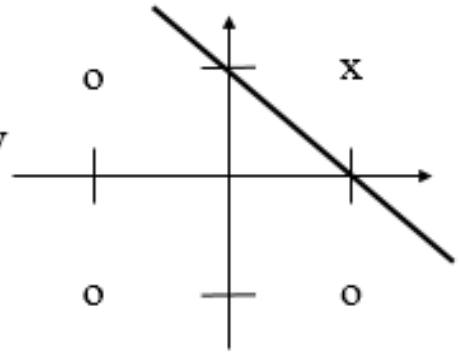
- Examples of linearly separable classes

  - Logical **AND** function

    patterns (bipolar) decision boundary

x1	x2	y
-1	-1	-1
-1	1	-1
1	-1	-1
1	1	1

w1 = 1
w2 = 1
b = -1
θ = 0
-1 + x1 + x2 = 0



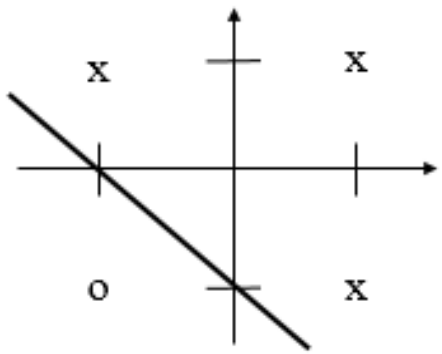
x: class I (y = 1)
o: class II (y = -1)

  - Logical **OR** function

    patterns (bipolar) decision boundary

x1	x2	y
-1	-1	-1
-1	1	1
1	-1	1
1	1	1

w1 = 1
w2 = 1
b = 1
θ = 0
1 + x1 + x2 = 0

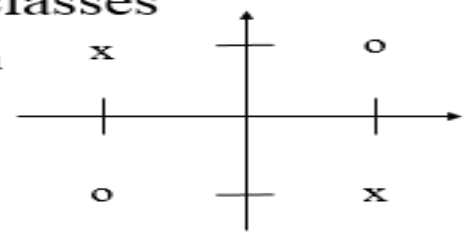


x: class I (y = 1)
o: class II (y = -1)

- Examples of linearly inseparable classes
  - Logical **XOR** (exclusive OR) function

patterns (bipolar) decision boundary

x1	x2	y
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1



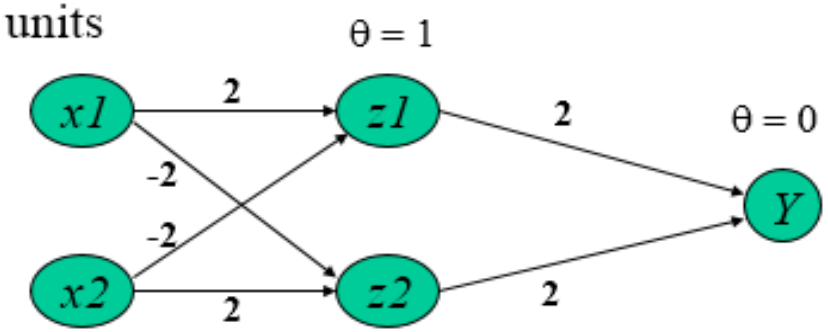
x: class I (y = 1)
o: class II (y = -1)

No line can separate these two classes, as can be seen from the fact that the following linear inequality system has no solution

$$\begin{cases} b - w_1 - w_2 < 0 & (1) \\ b - w_1 + w_2 \geq 0 & (2) \\ b + w_1 - w_2 \geq 0 & (3) \\ b + w_1 + w_2 < 0 & (4) \end{cases}$$

because we have b < 0 from (1) + (4), and b >= 0 from (2) + (3), which is a contradiction

– XOR can be solved by a more complex network with hidden units



(-1, -1)	(-1, -1)	-1
(-1, 1)	(-1, 1)	1
(1, -1)	(1, -1)	1
(1, 1)	(1, 1)	-1

Activation Function

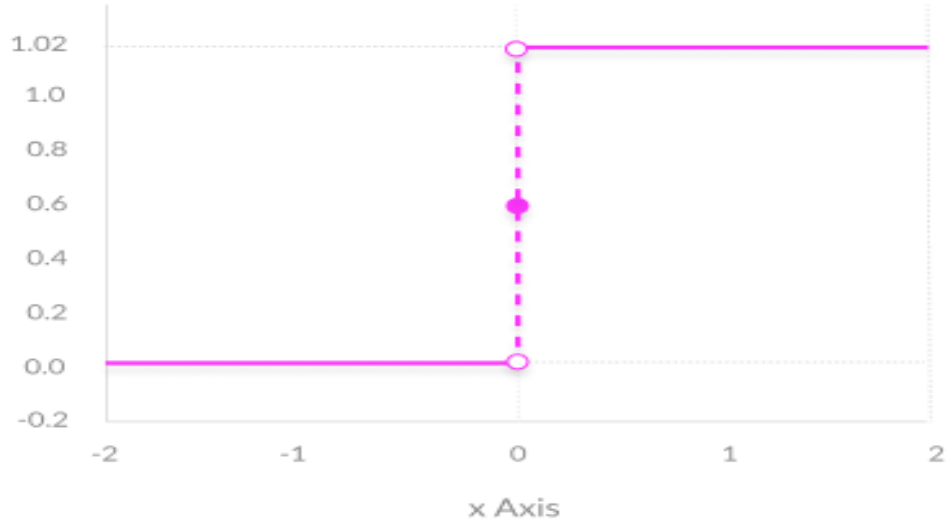
- **What** are **Activation functions** and what are it uses in a Neural Network Model?
- Activation functions are really important for a Artificial Neural Network to learn and make sense of something really complicated and Non-linear complex functional mappings between the inputs and response variable.
- It is also known as Transfer Function.
- Activation functions are important for a Artificial Neural Network to learn and understand the complex patterns.
- The main function of it is to introduce non-linear properties into the network.
- **What it does?**
- Their main purpose is to **convert** a input signal of a node in a A-NN to an output signal. That output signal now is used as a input in the next layer in the stack.
- The non linear activation function will help the model to understand the complexity and give accurate results.

- **why can’t we do it without activating the input signal?**
- If we do not apply a Activation function then the output signal would simply be a simple linear function.
- A linear function is just a polynomial of one degree. Now, a linear equation is easy to solve but they are limited in their complexity and have less power to learn complex functional mappings from data.
- A Neural Network without Activation function would simply be a Linear regression Model, which has limited power and does not performs good most of the times.
- Also without activation function our Neural network would not be able to learn and model other complicated kinds of data such as images, videos , audio , speech etc.

3-Types of Activation Functions

Binary Step Function

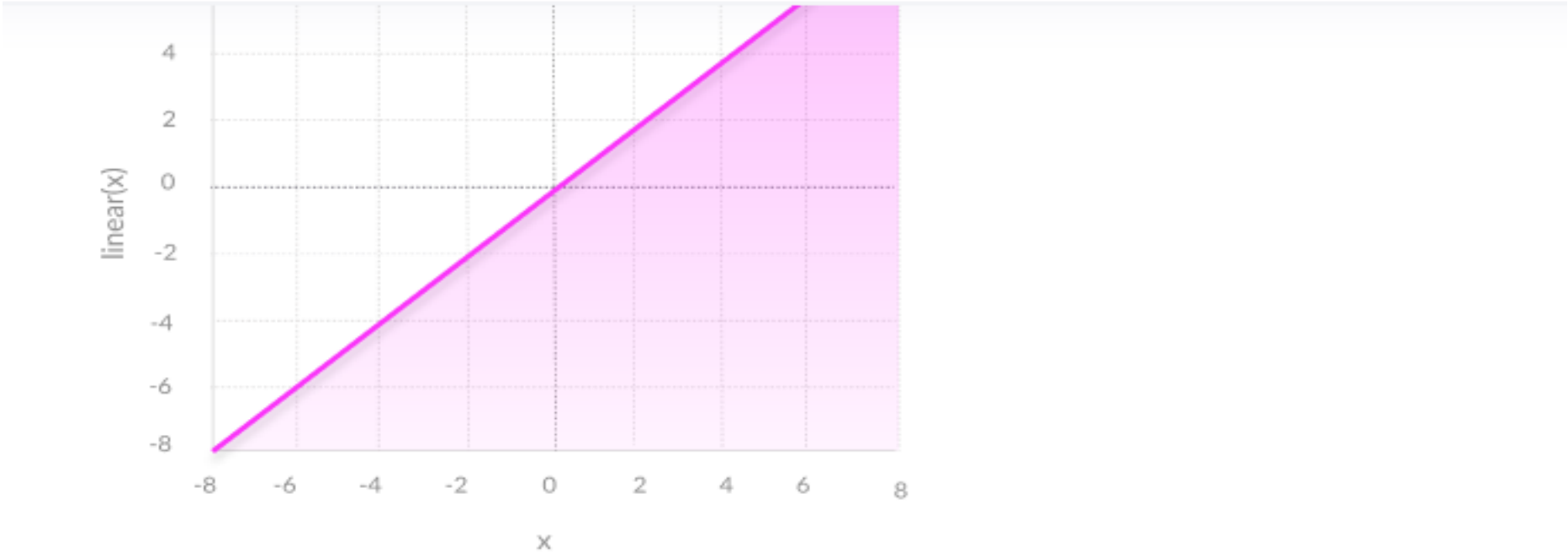
- A **binary step function** is a threshold-based activation function. If the input value is above or below a certain threshold, the neuron is activated and sends exactly the same signal to the next layer.



The problem with a step function is that it does not allow multi-value outputs—for example, it cannot support classifying the inputs into one of several categories.

- **Linear Activation Function**

A linear activation function takes the form: **A = cx**



It takes the inputs, multiplied by the weights for each neuron, and creates an output signal proportional to the input. In one sense, a linear function is better than a step function because it allows multiple outputs, not just yes and no.

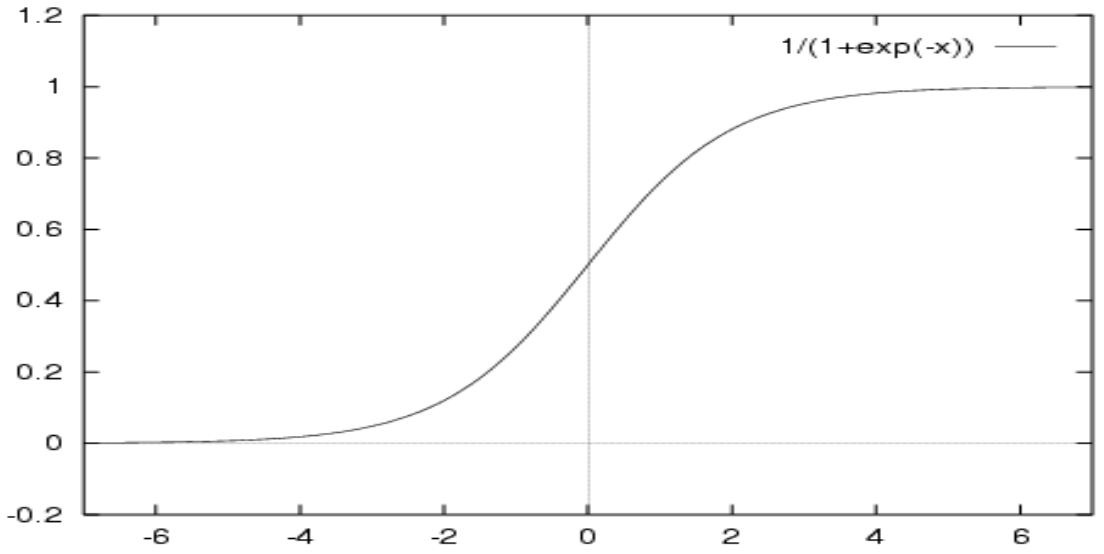
Non-Linear Activation Functions

- Non-linear functions are those which have degree more than one and they have a curvature.
- when we plot a Non-Linear function. Now we need a Neural Network Model to learn and represent almost anything and any arbitrary complex function which maps inputs to outputs.
- Non linear techniques are ANN and SVM.
- Modern neural network models use non-linear activation functions.
- They allow the model to create complex mappings between the network's inputs and outputs, which are essential for learning and modeling complex data, such as images, video, audio, and data sets which are non-linear or have high dimensionality.
- Almost any process imaginable can be represented as a functional computation in a neural network, provided that the activation function is non-linear.
- Non-linear functions address the problems of a linear activation function: They allow backpropagation because they have a derivative function which is related to the inputs.
- They allow "stacking" of multiple layers of neurons to create a deep neural network. Multiple hidden layers of neurons are needed to learn complex data sets with high levels of accuracy.
- **Most popular types of Activation functions -**

- Sigmoid or Logistic
- Hyperbolic tangent
- ReLu -Rectified linear units

Sigmoid Activation function

- It is a activation function of form **f(x) = 1 / 1 + exp(-x) .**
- Its Range is between 0 and 1. It is a S — shaped curve.
- It is easy to understand and apply but it has major reasons which have made it fall out of popularity -
- Vanishing gradient problem
- Secondly , its output isn't zero centered. It makes the gradient updates go too far in different directions. **0 < output < 1, and it makes optimization harder.**
- Sigmoids saturate and kill gradients.
- Sigmoids have slow convergence.



Sigmoid / Logistic

Advantages

- **Smooth gradient**, preventing “jumps” in output values.
- **Output values bound** between 0 and 1, normalizing the output of each neuron.
- **Clear predictions**—For X above 2 or below -2, tends to bring the Y value (the prediction) to the edge of the curve, very close to 1 or 0. This enables clear predictions.

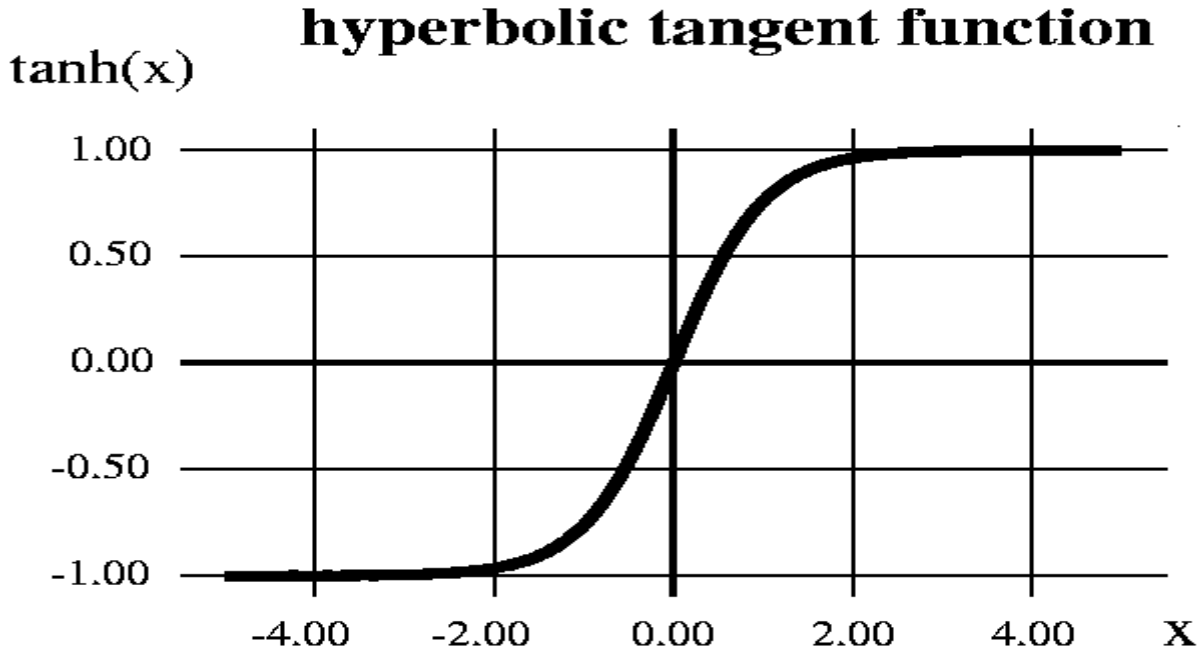
Disadvantages

- **Vanishing gradient**—for very high or very low values of X, there is almost no change to the prediction, causing a vanishing gradient problem. This can result in the network refusing to learn further, or being too slow to reach an accurate prediction.
- **Outputs not zero centered**.
- **Computationally expensive**

Hyperbolic Tangent function- Tanh

It’s mathamatical formula is **f(x) = 1 — exp(-2x) / 1 + exp(-2x).**

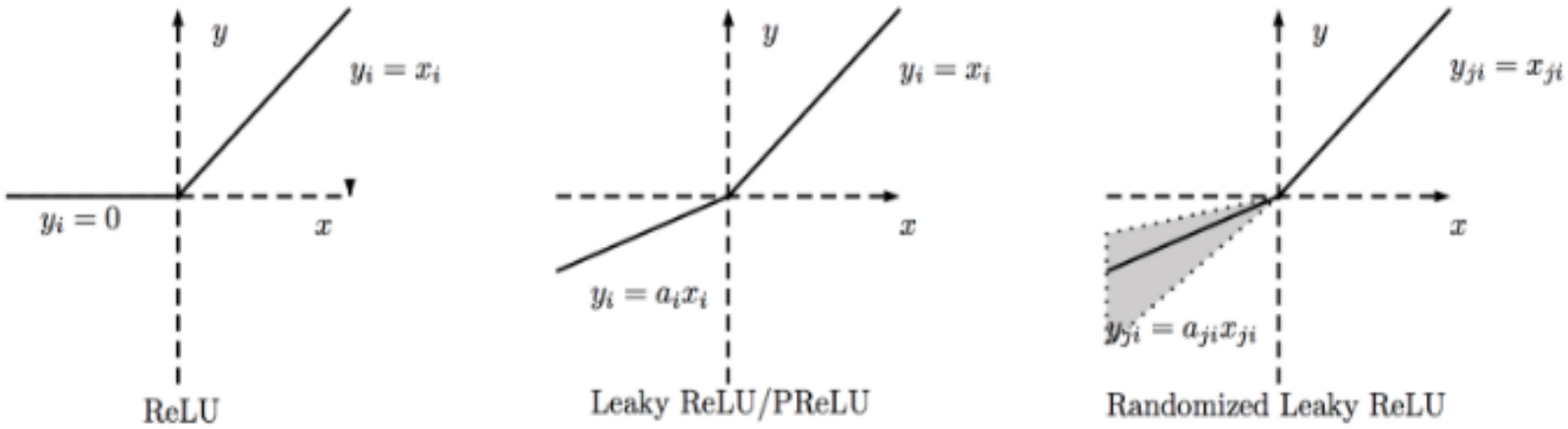
- Now it’s output is zero centered because its range in between -1 to 1
- ie -1 < output < 1 .
- Hence optimization is *easier* in this method hence in practice it is always preferred over Sigmoid function . But still it suffers from Vanishing gradient problem.



ReLu- Rectified Linear units

- It has become very popular due to 6 times improvement in convergence from Tanh function.
- It’s just R(x) = max(0,x) i.e if x < 0 , R(x) = 0 and if x >= 0 , R(x) = x.
- Hence as seeing the mathematical form of this function we can see that it is very simple and efficient .
- A lot of times in Machine learning and computer science we notice that most simple and consistent techniques and methods are only preferred and are best.
- Hence it **avoids and rectifies vanishing** gradient problem . Almost all deep learning Models use ReLu nowadays.
- But its **limitation** is that it should only be used **within Hidden layers** of a Neural Network Model.
- For output layers we should use a **Softmax** function for a Classification problem to compute the probabilities for the classes.
- For a regression problem it should simply use a **linear** function.

- Another problem with ReLu is that some gradients can be breakable during training and can die.
- It can cause a weight update which will makes it never activate on any data point again. Simply saying that ReLu could result in Dead Neurons.
- To fix this problem another modification was introduced called ***Leaky ReLu*** to fix the problem of dying neurons. It introduces a small slope to keep the updates alive.
- We then have another variant made form both ReLu and Leaky ReLu called **Maxout** function .

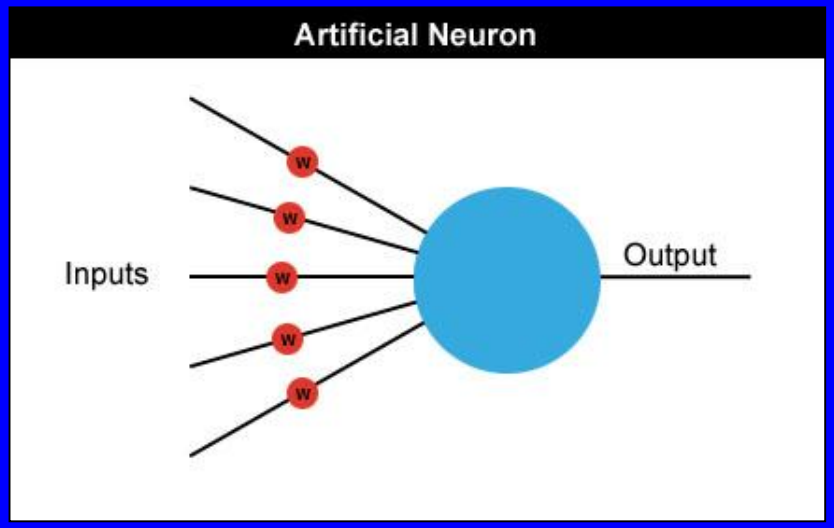


weights and bias

Neural networks

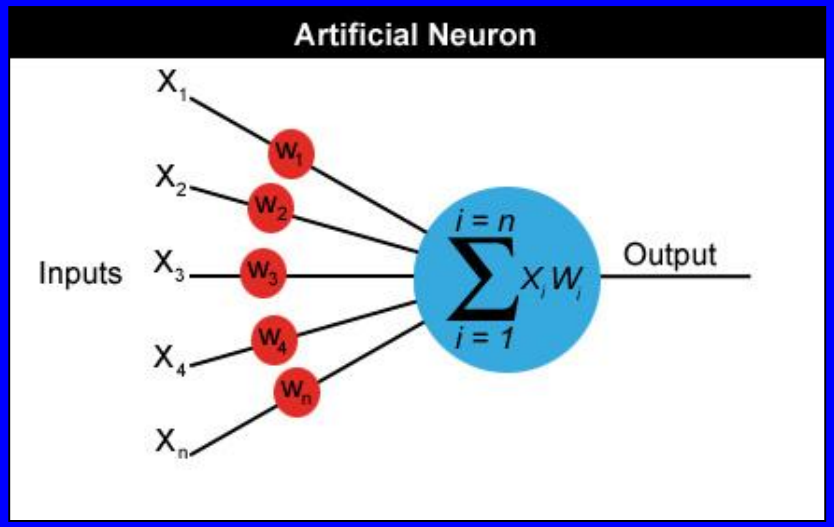
Neural networks

- Neural networks are made up of many artificial neurons.
- Each input into the neuron has its own weight associated with it illustrated by the red circle.
- A weight is simply a floating point number and it's these we adjust when we eventually come to train the network.



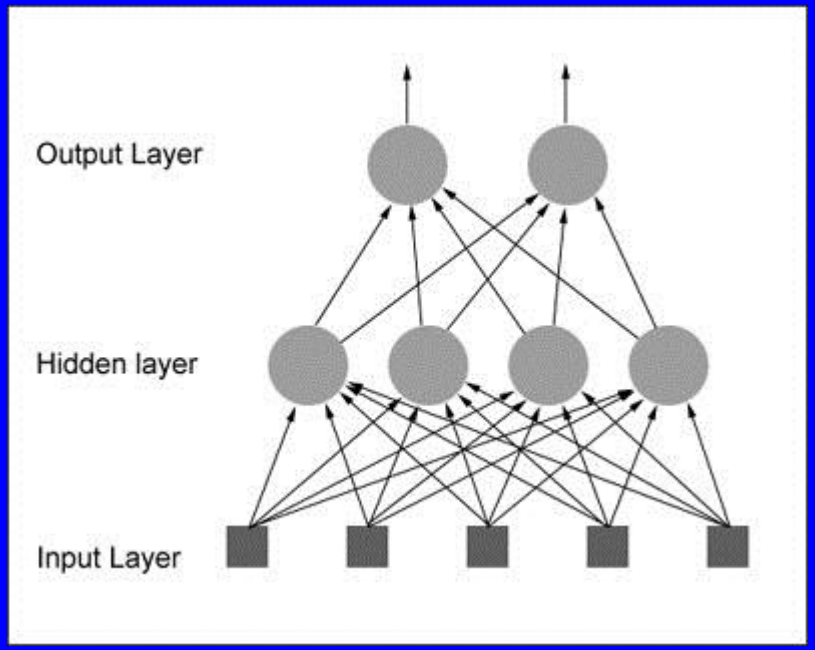
Neural networks

- A neuron can have any number of inputs from one to n, where n is the total number of inputs.
- The inputs may be represented therefore as x_1, x_2, x_3… x_n.
- And the corresponding weights for the inputs as w_1, w_2, w_3… w_n.
- Output $a = x_1w_1+x_2w_2+x_3w_3... +x_nw_n$

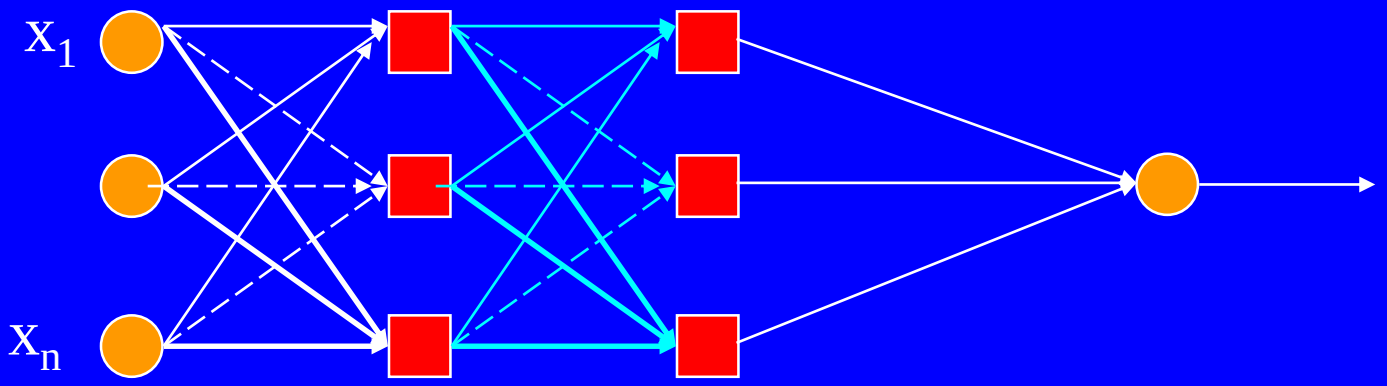


How do we actually *use* an artificial neuron?

- feedforward network: The neurons in each layer feed their output forward to the next layer until we get the final output from the neural network.
- There can be any number of hidden layers within a feedforward network.
- The number of neurons can be completely arbitrary.



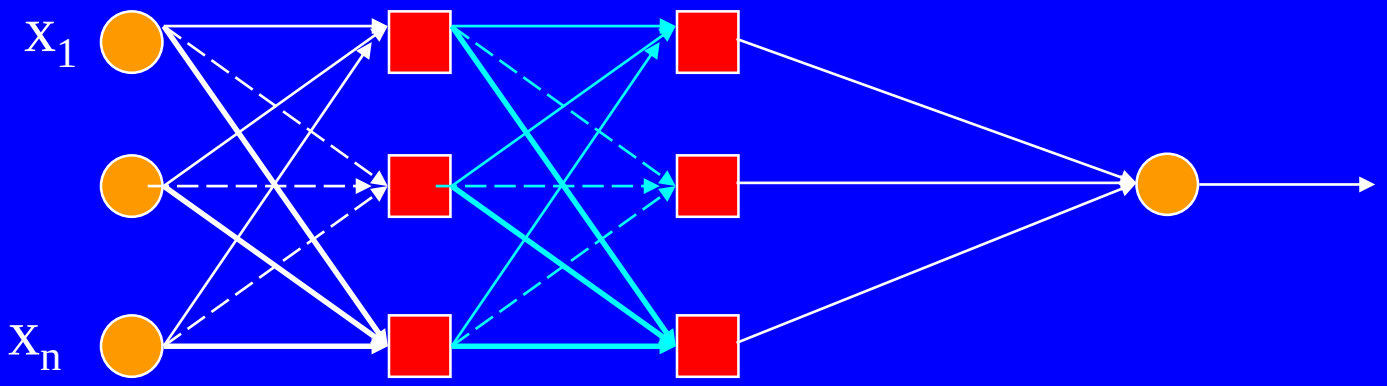
Multi-Layer Perceptron (MLP)



We will introduce the MLP and the backpropagation algorithm which is used to train it

MLP used to describe any general feedforward (no recurrent connections) network

However, we will concentrate on nets with units arranged in layers



NB different books refer to the above as either 4 layer (no. of layers of neurons) or 3 layer (no. of layers of adaptive weights). We will follow the latter convention

1st question:

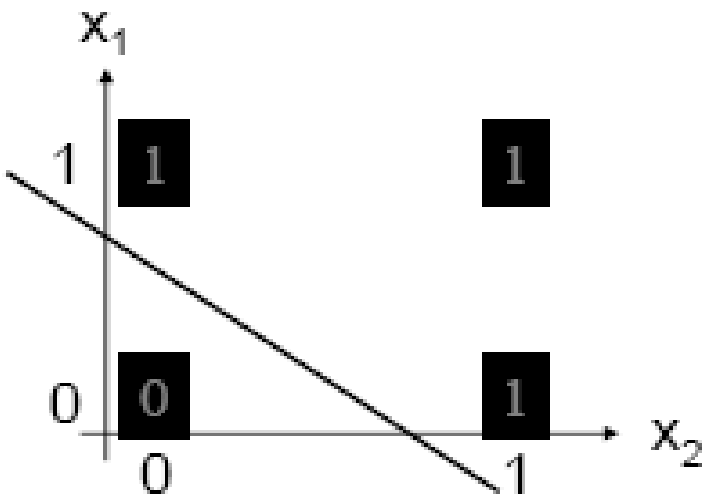
what do the extra layers gain you? Start with looking at what a single layer can't do

Perceptron Learning Theorem

- A perceptron (threshold unit) can *learn* anything that it can *represent* (i.e. anything separable with a hyperplane)

OR function

x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	1

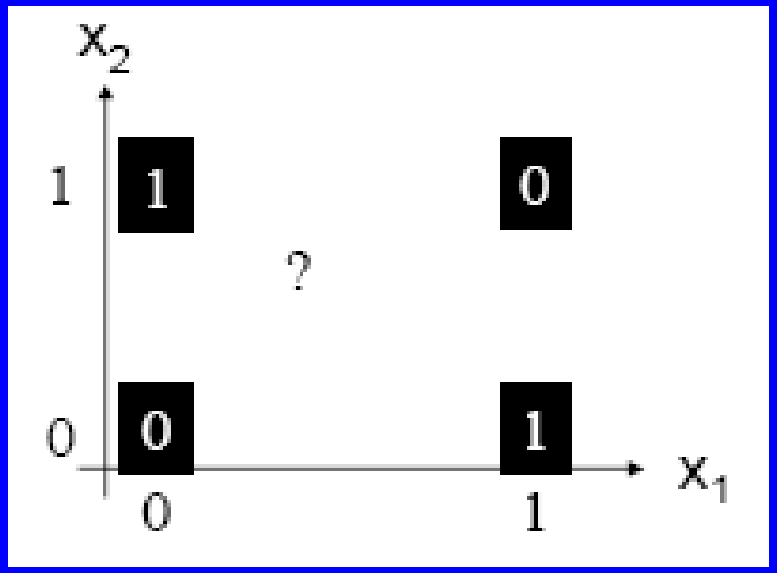


The Exclusive OR problem

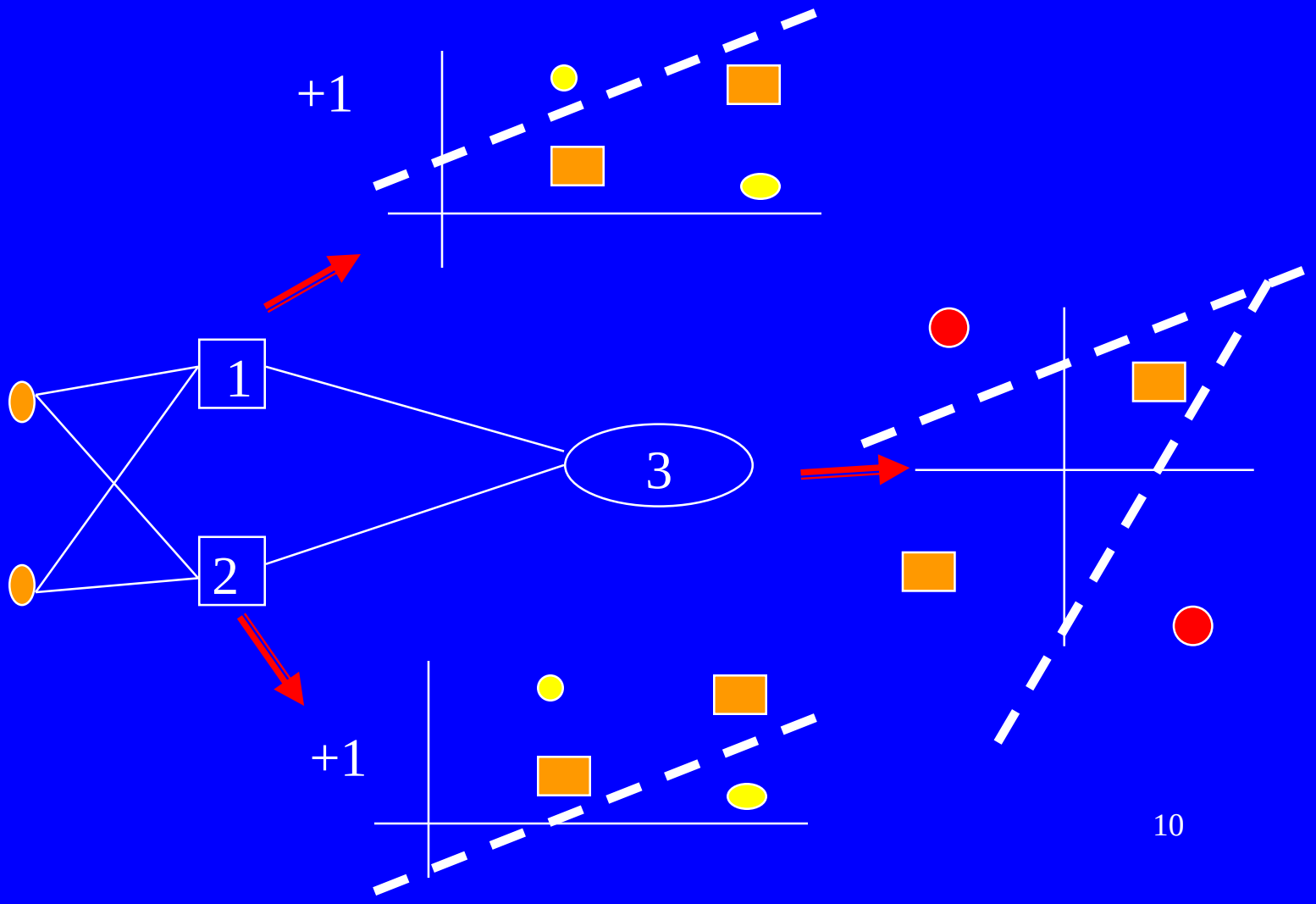
A Perceptron cannot represent Exclusive OR since it is not linearly separable.

XOR function

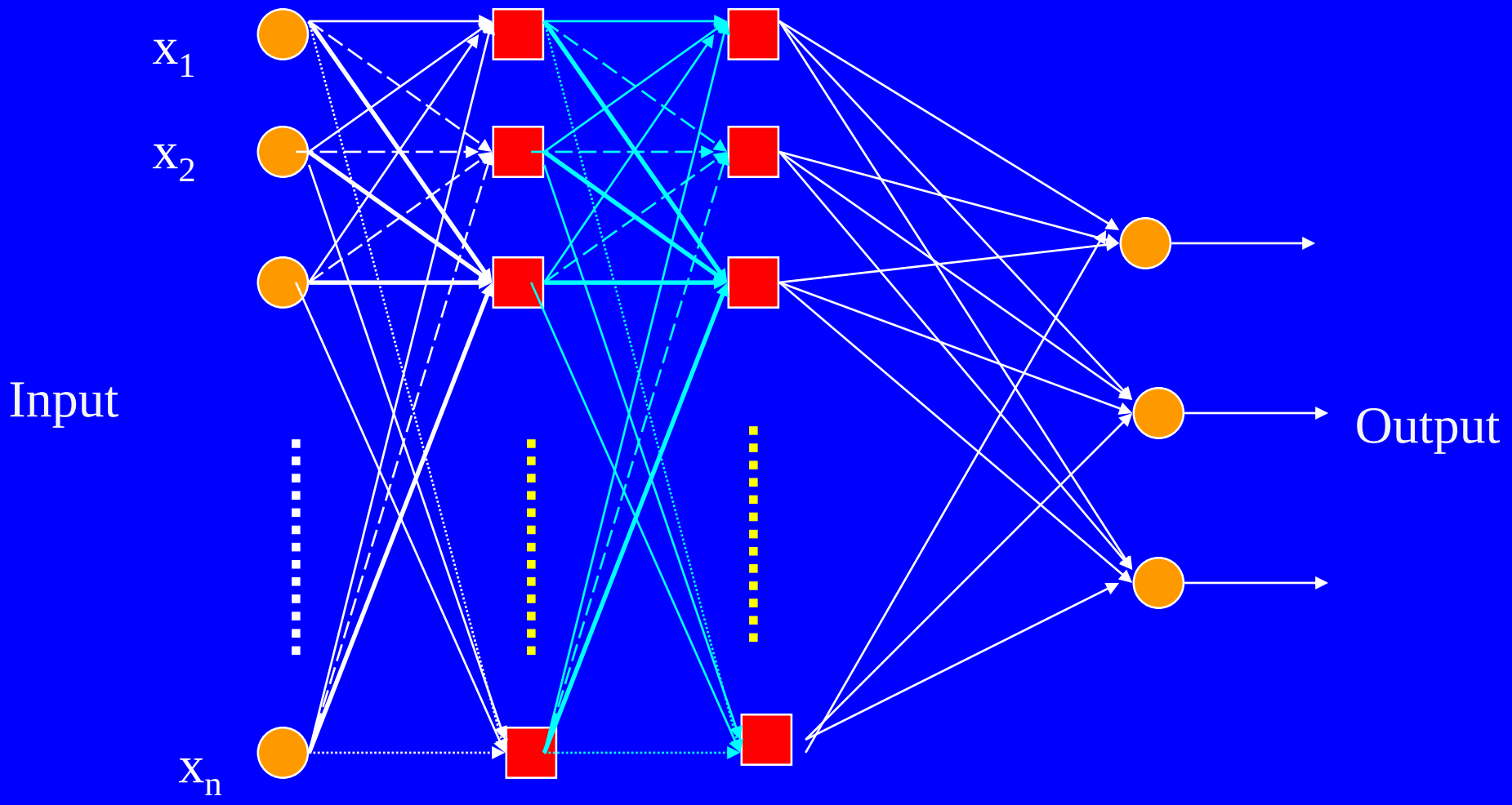
x_1	x_2	y
0	0	0
0	1	1
1	0	1
1	1	0



Minsky & Papert (1969) offered solution to XOR problem by combining perceptron unit responses using a second layer of Units. Piecewise linear classification using an MLP with threshold (perceptron) units

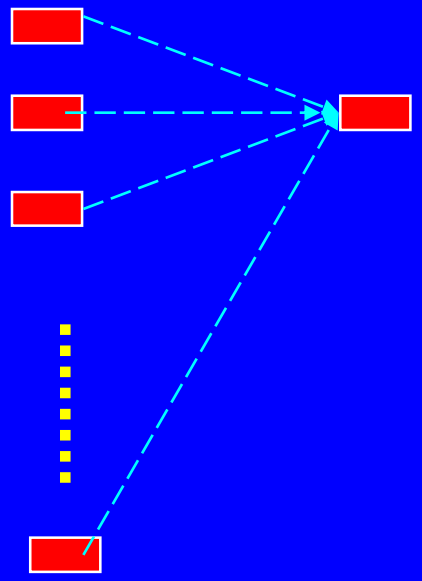


Three-layer networks



Properties of MLP architecture

- No connections within a layer
- No direct connections between input and output layers
- Fully connected between layers
- Often more than 3 layers
- Number of output units need not equal number of input units
- Number of hidden units per layer can be more or less than input or output units

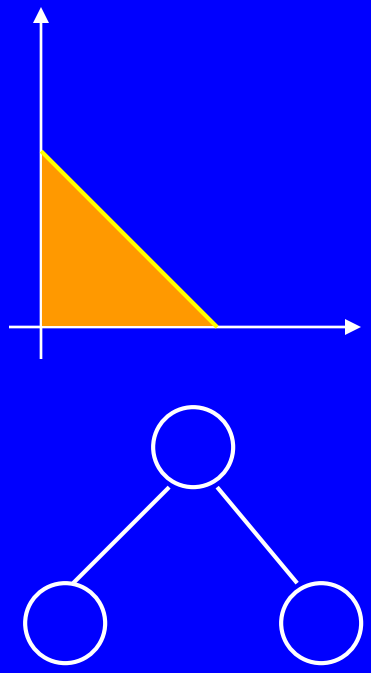


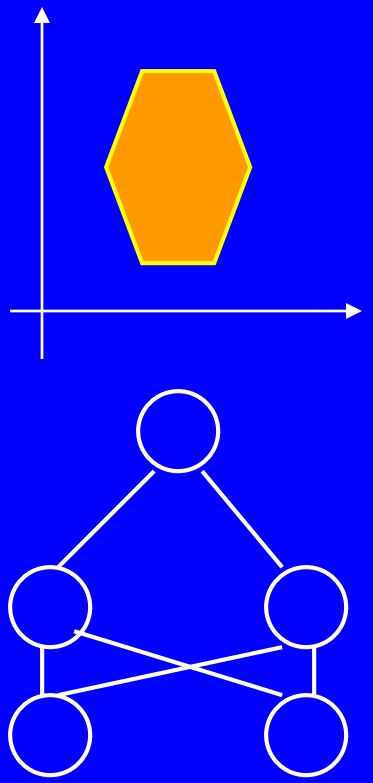
Each unit is a perceptron

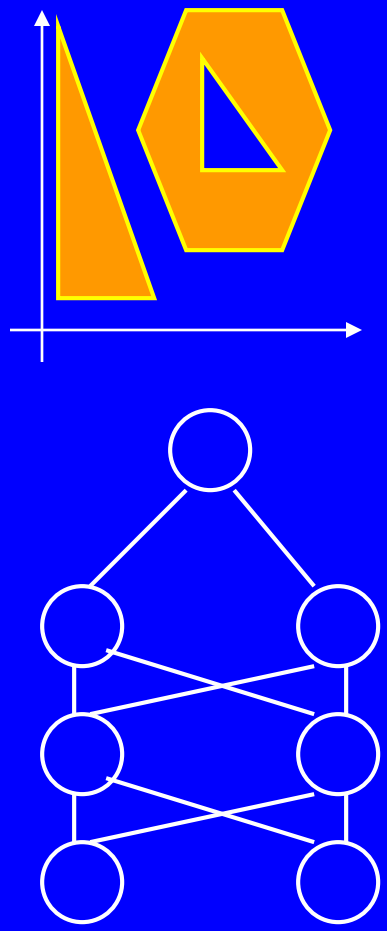
$$y_i = f\left(\sum_{j=1}^{m} w_{ij} x_j + b_i\right)$$

Often include bias as an extra weight

What do each of the layers do?







1st layer draws linear boundaries

2nd layer combines the boundaries

3rd layer can generate arbitrarily complex boundaries

Backpropagation learning algorithm 'BP'

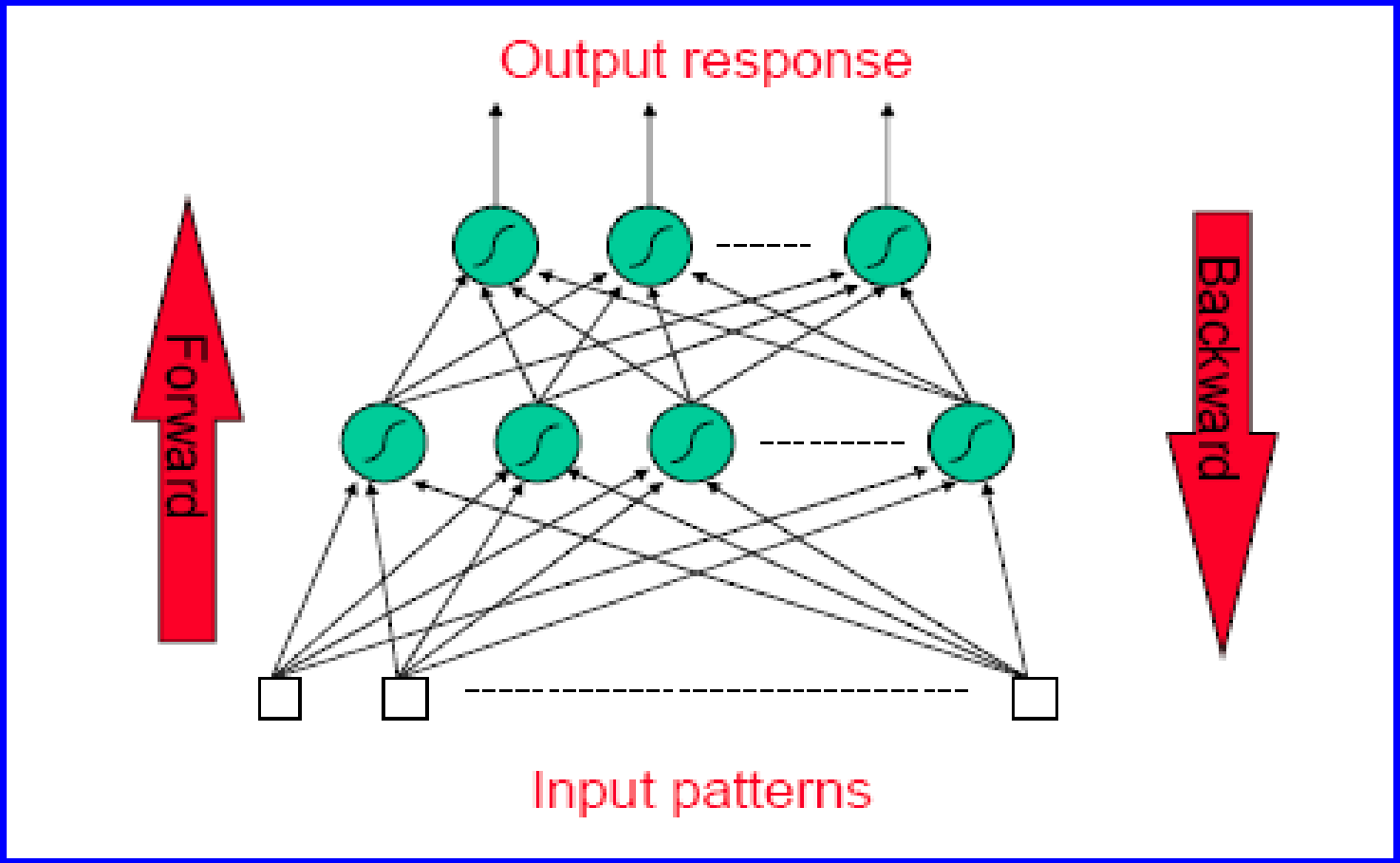
Solution to credit assignment problem in MLP. *Rumelhart, Hinton and Williams (1986) (*though actually invented earlier in a PhD thesis relating to economics)

BP has two phases:

Forward pass phase: computes 'functional signal', feed forward propagation of input pattern signals through network

Backward pass phase: computes 'error signal', *propagates* the error *backwards* through network starting at output units (where the error is the difference between actual and desired output values)

Conceptually: Forward Activity - Backward Error



Forward Propagation of Activity

- Step 1: Initialise weights at random, choose a learning rate η
- Until network is trained:
- For each training example i.e. input pattern and target output(s):
- Step 2: Do forward pass through net (with fixed weights) to produce output(s)
  - i.e., in Forward Direction, layer by layer:
    - Inputs applied
    - Multiplied by weights
    - Summed
    - 'Squashed' by sigmoid activation function
    - Output passed to each neuron in next layer
  - Repeat above until network output(s) produced

Step 3. Back-propagation of error

• Compute error (delta or local gradient) for each output unit δ_k
• Layer-by-layer, compute error (delta or local gradient) for each hidden unit δ_j by backpropagating errors (as shown previously)

Step 4: Next, update all the weights Δw_{ij}
By gradient descent, and go back to Step 2

– The overall MLP learning algorithm, involving forward pass and backpropagation of error (until the network training completion), is known as the Generalised Delta Rule (GDR), or more commonly, the Back Propagation (BP) algorithm

'Back-prop' algorithm summary

(with Maths!) (Not Examinable)

- Initialise weights at random, choose a learning rate η
- Until network is trained:
  - For each training example (input pattern and target outputs):
  - Do forward pass through net (with fixed weights) to produce outputs
    -assuming J hidden layer nodes and N inputs for a 2-layer MLP:
    $y_k = f(\sum_{j=0} w_{jk} o_j)$ where o_j is output from each hidden node j: $o_j = f\left(\sum_{i=0}^{N} w_{ij} x_i\right)$
  - For each output unit k, compute deltas: $\delta_k = (y_{target_k} - y_k) y_k (1 - y_k)$
  - For hidden units j (from last to first hidden layer, for the case of more than 1 hidden layer) compute deltas: $\delta_j = o_j (1 - o_j) \sum_k w_{jk} \delta_k$
  - For all weights, change weight by gradient descent: $\Delta w_{ij} = \eta \delta_j y_i$
    -Specifically, for the 2-layer MLP, for weight from input layer unit i to hidden layer unit j, the weight changes by: $\Delta w_{ij} = \eta \delta_j x_i$
    And, for weight from hidden layer unit j to output layer unit k, weight changes $\Delta w_{jk} = \eta \delta_k o_j$

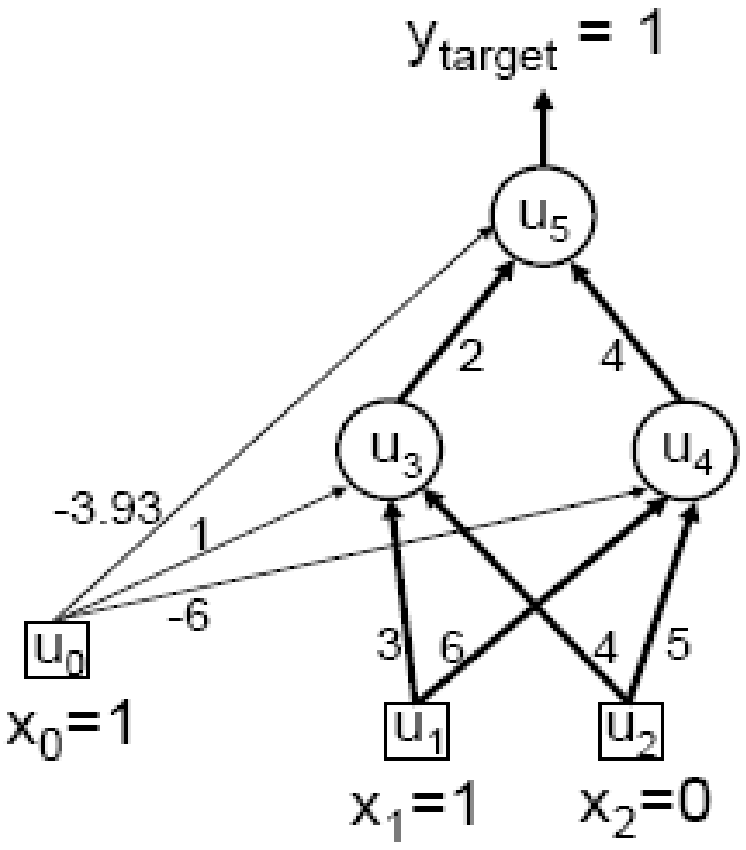
‘Back-prop’ algorithm summary

(with NO Maths!)

- Initialise weights at random, choose a learning rate η
- Until network is trained:
  - For each training example (input pattern and target outputs):
    - Do forward pass through net (with fixed weights) to produce network outputs
    - For each output unit *k*, compute deltas (local gradients):
    - For hidden units *j* (from last to first hidden layer, for the case of more than 1 hidden layer) compute deltas (local gradients):
    - For all weights, change weight by gradient descent (generalized Delta Rule): $\Delta w_{jk} = \eta \delta_k o_j$

      where o_j is the input to the neuron *k*, and η is the learning rate, and δ_k is local gradient for the neuron

MLP/BP: A worked example



Current state:

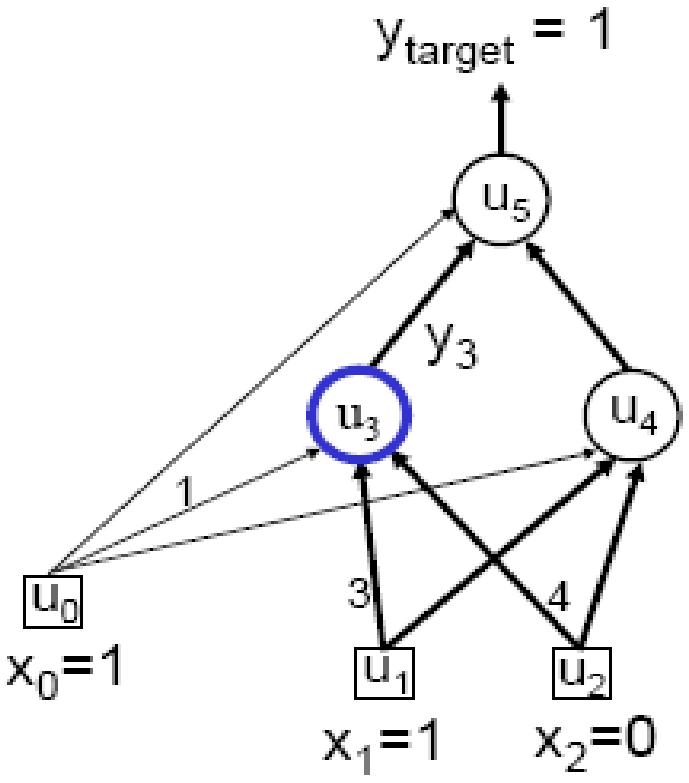
- Weights on arrows e.g. $w_{13} = 3$, $w_{35} = 2$, $w_{24} = 5$
- Bias weights, e.g.

bias for unit 4 (u_4) is $w_{04} = -6$

Training example (e.g. for logical OR problem):

- Input pattern is $x_1=1$, $x_2=0$
- Target output is $y_{target}=1$

Worked example: Forward Pass



Output for any neuron/unit j can be calculated from:

$$a_j = \sum_i w_{ij} x_i$$

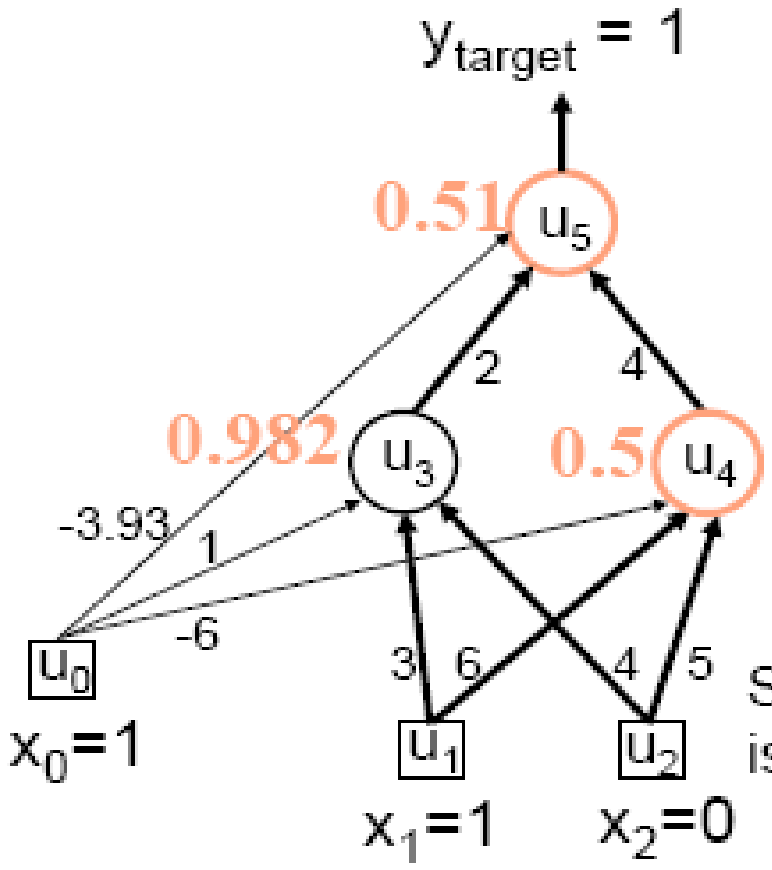
$$y_j = f(a_j) = \frac{1}{1+e^{-a_j}}$$

e.g Calculating output for Neuron/unit 3 in hidden layer:

$a_3 = 1*1 + 3*1 + 4*0 = 4$

$y_3 = f(4) = \frac{1}{1+e^{-4}} = 0.982$

Worked example: Forward Pass

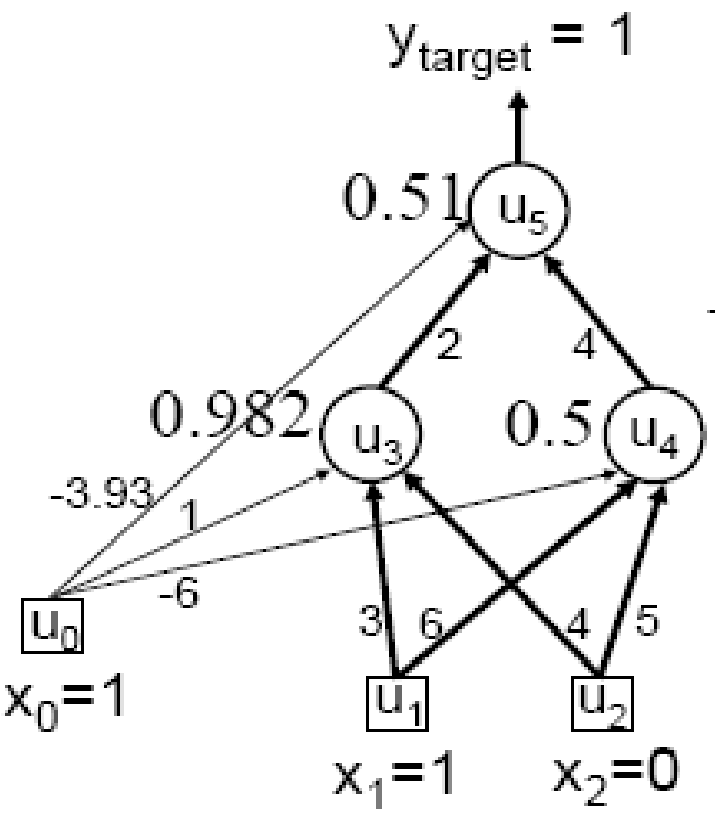


Unit	activation a_j	output y_j
u_3	4.00	0.982
u_4	0.00	0.500
u_5	0.04	**0.510**

(network output)

So the error for this training example is: $(y_{target} - y_5) = (1 - 0.510) = 0.490$

Worked example: Backward Pass



Now compute delta values starting at the output:

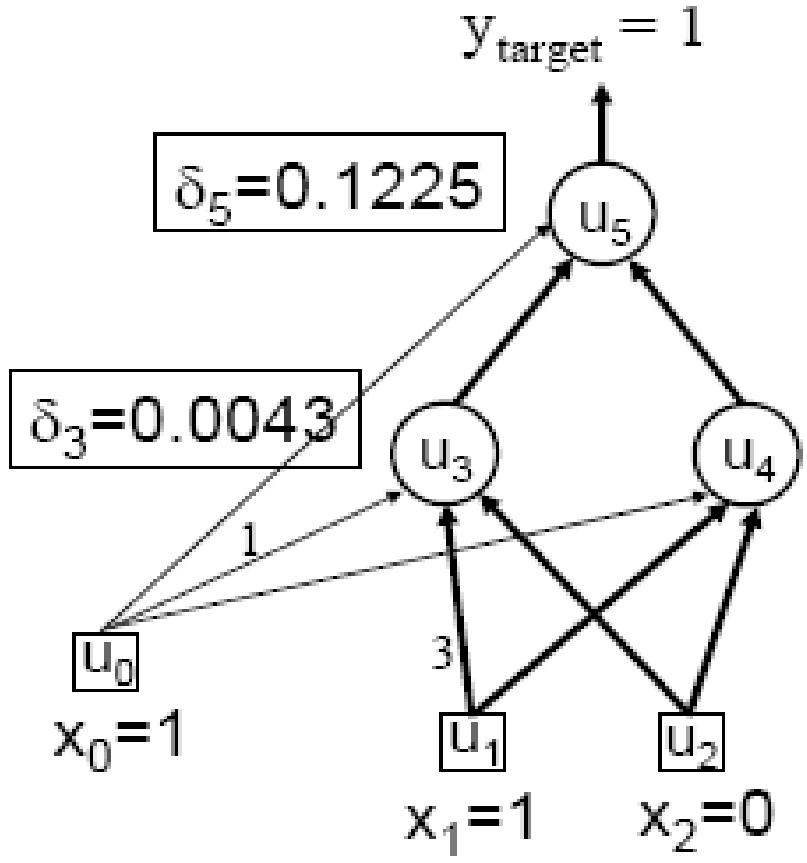
δ_5 $= y_5(1 - y_5)\,(y_{target} - y_5)$
$= 0.51(1 - 0.51) \times 0.49$
$=$ **0.1225**

Then for hidden units:

δ_4 $= y_4(1 - y_4)\, w_{45}\, \delta_5$
$= 0.5(1 - 0.5) \times 4 \times 0.1225$
$=$ **0.1225**

δ_3 $= y_3(1 - y_3)\, w_{35}\, \delta_5$
$= 0.982(1-0.982) \times 2 \times 0.1225$
$=$ **0.0043**

Worked example: Update Weights Using Generalized Delta Rule (BP)



- Set learning rate $\eta = 0.1$

Change weights by:

$$\Delta w_{ij} = \eta \delta_j y_i$$

- e.g. bias weight on u_3:

$\Delta w_{03} = \eta \delta_3 x_0$
$=0.1*0.0043*1$
$= 0.0004$

So, new w_{03} ⌫
$w_{03}(old)+\Delta w_{03}$
$=1+0.0004=1.0004$

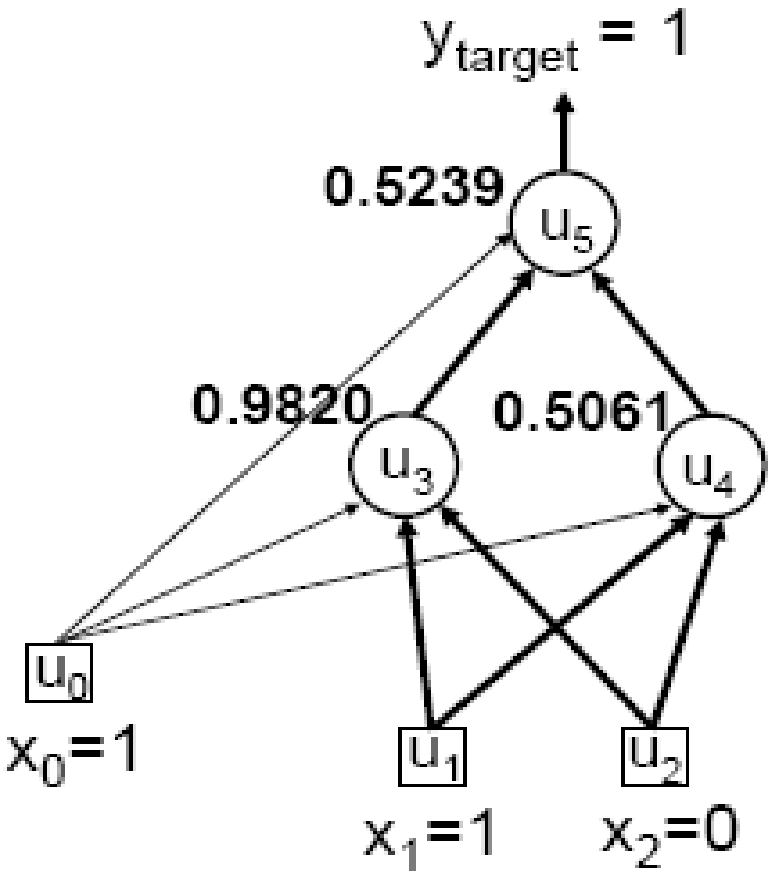
- and likewise:

w_{13} ⌫ 3 + 0.0004
= 3.0004

Similarly for the all weights wij:

i	j	w_{ij}	δ_j	y_i	Updated w_{ij}
0	3	**1**	0.0043	1.0	**1.0004**
1	3	**3**	0.0043	1.0	**3.0004**
2	3	**4**	0.0043	0.0	**4.0000**
0	4	**-6**	0.1225	1.0	**-5.9878**
1	4	**6**	0.1225	1.0	**6.0123**
2	4	**5**	0.1225	0.0	**5.0000**
0	5	**-3.92**	0.1225	1.0	**-3.9078**
3	5	**2**	0.1225	0.9820	**2.0120**
4	5	**4**	0.1225	0.5	**4.0061**

Verification that it works



On next forward pass:

The new activations are:

$y_3 = f(4.0008) = 0.9820$

$y_4 = f(0.0245) = 0.5061$

$y_5 = f(0.0955) = \mathbf{0.5239}$

Thus the new error

$(y_{target} - y_5) = (1 - 0.5239) = 0.476$

has been reduced by **0.014**

(from **0.490** to **0.476**)

Ref: "Neural Network Learning & Expert Systems" by Stephen Gallant

Training

- This was a single iteration of back-prop
- Training requires many iterations with many training examples or *epochs* (one epoch is entire presentation of complete training set)
- It can be slow !
- Note that computation in MLP is local (with respect to each neuron)
- Parallel computation implementation is also possible

Training and testing data

- How many examples ?
  - The more the merrier !
- Disjoint training and testing data sets
  - learn from training data but evaluate performance (generalization ability) on unseen test data
- **Aim**: minimize error on *test* data

weights and bias

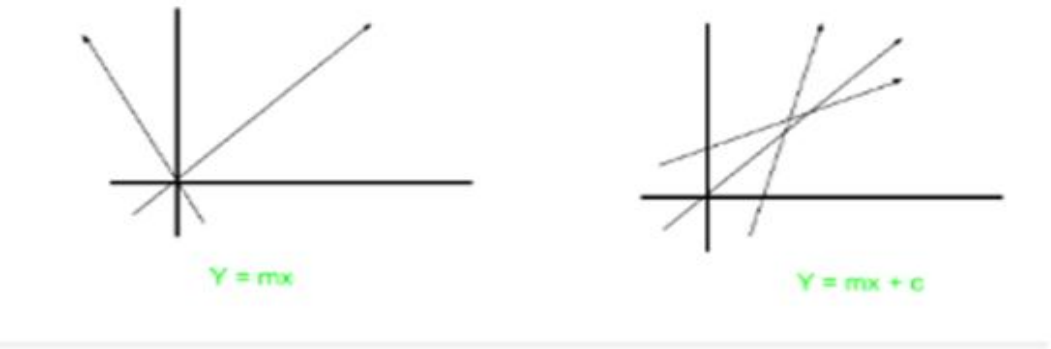
- Bias is just like an intercept added in a linear equation. It is an additional parameter in the Neural Network which is used to adjust the output along with the weighted sum of the inputs to the neuron.
- bias value allows you to shift the activation function to either right or left.

output = sum (weights * inputs) + bias

- The output is calculated by multiplying the inputs with their weights and then passing it through an activation function like the Sigmoid function, etc. Here, bias acts like a constant which helps the model to fit the given data. The steepness of the Sigmoid depends on the weight of the inputs.

It allows you to move the line down and up fitting the prediction with the data better.

- **If bias=0 ie ,** the constant c is absent then the line will pass through the origin (0, 0) and you will get a poorer fit. (Neural Network is an important component of Machine Learning)



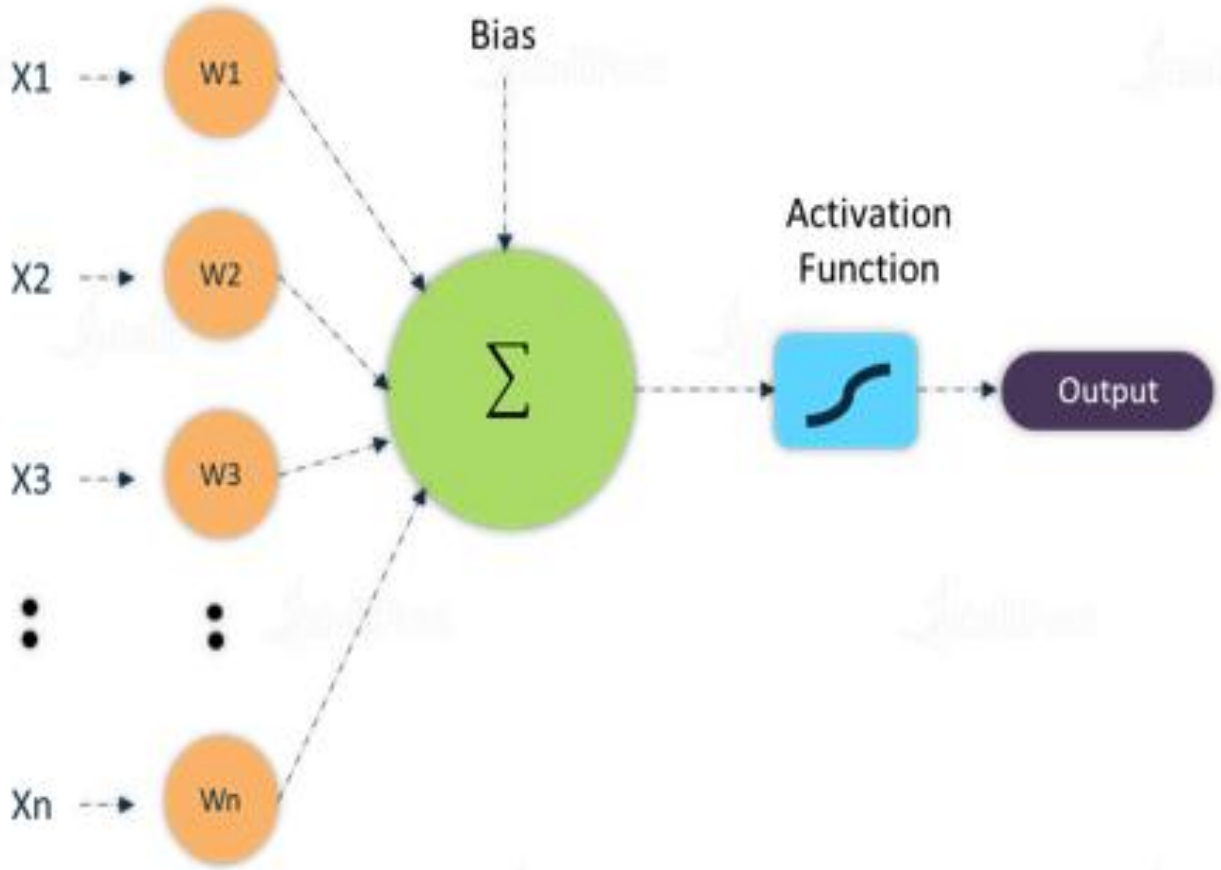
- If the inputs are x1,x2,x3, then the weight applied to them are denoted as w1,w2,w3. Then the output is as follows: $y = f(x) = \Sigma x_i w_i$
- Bias is like the intercept added in a linear equation. It is an additional parameter in the Neural Network which is used to adjust the output along with the weighted sum of the inputs to the neuron. Thus, Bias is a constant which helps the model in a way that it can fit best for the given data.
- The processing done by the neuron is:

*output = sum (weights * inputs) + bias*

For example, consider an equation y=mx+c

Here m is acting as weight and the constant c is acting as bias.

- Bias units are not connected to any previous layer, it is just appended to the start/end of the input and each hidden layer, and is not affected by the values in the previous layer. Explain with diagram:



Schematic Representation of a Neuron in a Neural Network

Here x1,x2,x3 are inputs and w1, w2,w3 are weights. It takes an input, processes it, passes it through an activation function, and returns the output.

Loss function

- If predictions deviates too much from actual results.
- It improve with the help of some optimization function, loss function learns to reduce the error in prediction.

Type of loss functions: It can be classified into two major categories.

Regression losses and **Classification losses**.

Classification: we are trying to predict output from set of finite categorical values i.e Given large data set of images of hand written digits, categorizing them into one of 0–9 digits.

Regression, on the other hand, deals with predicting a continuous value for example given floor area, number of rooms, size of rooms, predict the price of room.

n - Number of training examples.

i - ith training example in a data set.

y(i) - Ground truth label for ith training example.

y_hat(i) - Prediction for ith training example.

- **Regression Losses:**
- **Mean Square Error/Quadratic Loss/L2 Loss**
- ***Mathematical formulation*** :-

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

- *Mean square error* is measured as the average of squared difference between predictions and actual observations.
- Due to squaring, predictions which are far away from actual values are penalized heavily in comparison to less deviated predictions.
- MSE has nice mathematical properties which makes it easier to calculate gradients.

- **Mean Absolute Error/L1 Loss**
- ***Mathematical formulation*** :-

$$MAE = \frac{\sum_{i=1}^{n} | y_i - \hat{y}_i |}{n}$$

- ***Mean absolute error*****:** It is measured as the average of sum of absolute differences between predictions and actual observations.
- This as well measures the magnitude of error without considering their direction.
- MAE needs more complicated tools such as linear programming to compute the gradients. (MAE is more robust to **outliers** since it does not make use of square)
- **Mean Bias Error:** This is same as MSE, with the only difference that we don't take absolute values, it could determine if the model has positive bias or negative bias.

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

- **Classification Losses: (Multi class SVM Loss)**
- It is used for maximum-margin classification, most notably for support vector machines. Although not differentiable, it's a convex function which makes it easy to work with usual convex optimizers used in machine learning domain.

$$SVMLoss = \sum_{j \neq y_i} max(0, s_j - s_{y_i} + 1)$$

Consider an example where we have three training examples and three classes to predict — Dog, cat and horse. Below the values predicted by our algorithm for each of the classes :

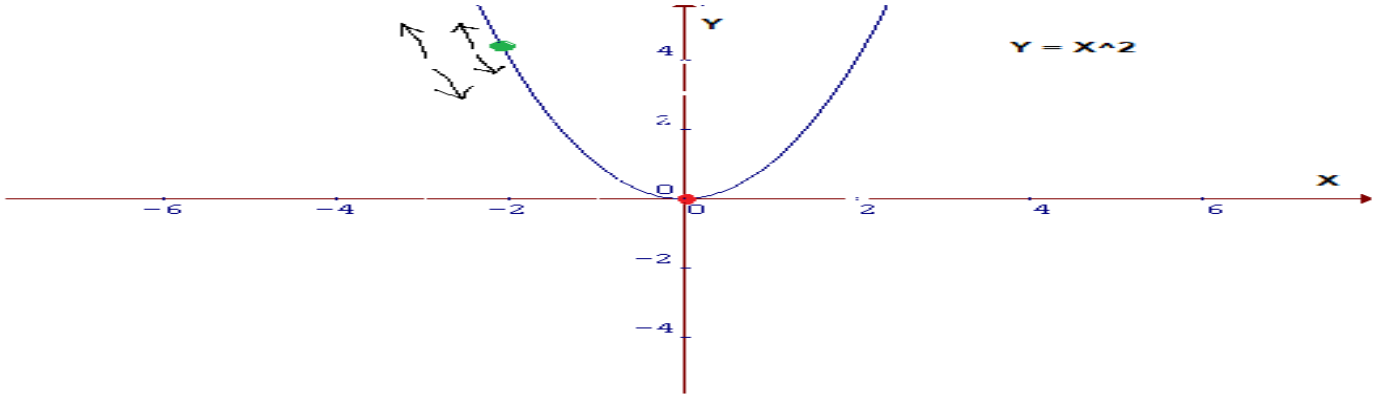


	Image #1	Image #2	Image #3
Dog	-0.39	-4.61	1.03
Cat	1.49	3.28	-2.37
Horse	4.21	1.46	-2.27

Gradient Descent Algorithm (GDA)

- **Gradient descent** is one of the most popular algorithms to perform optimization and by far the most common way to optimize neural networks. It is an iterative optimization algorithm used to find the minimum value for a function.

Consider that you are walking along the graph below, and you are currently at the '**green**' dot. Your aim is to reach the minimum i.e the 'red' dot, but from your position, you are unable to view it. Why we need of GDA? (To reduce cost function of higher degree.)

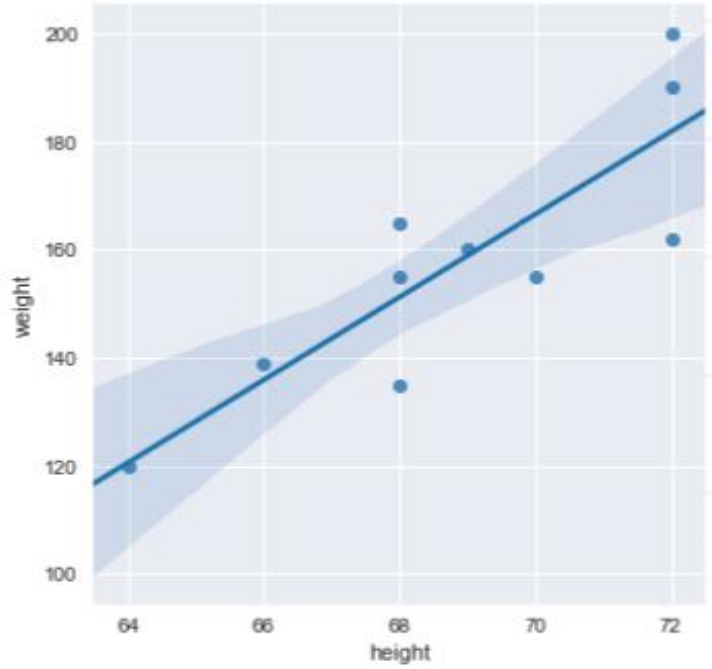


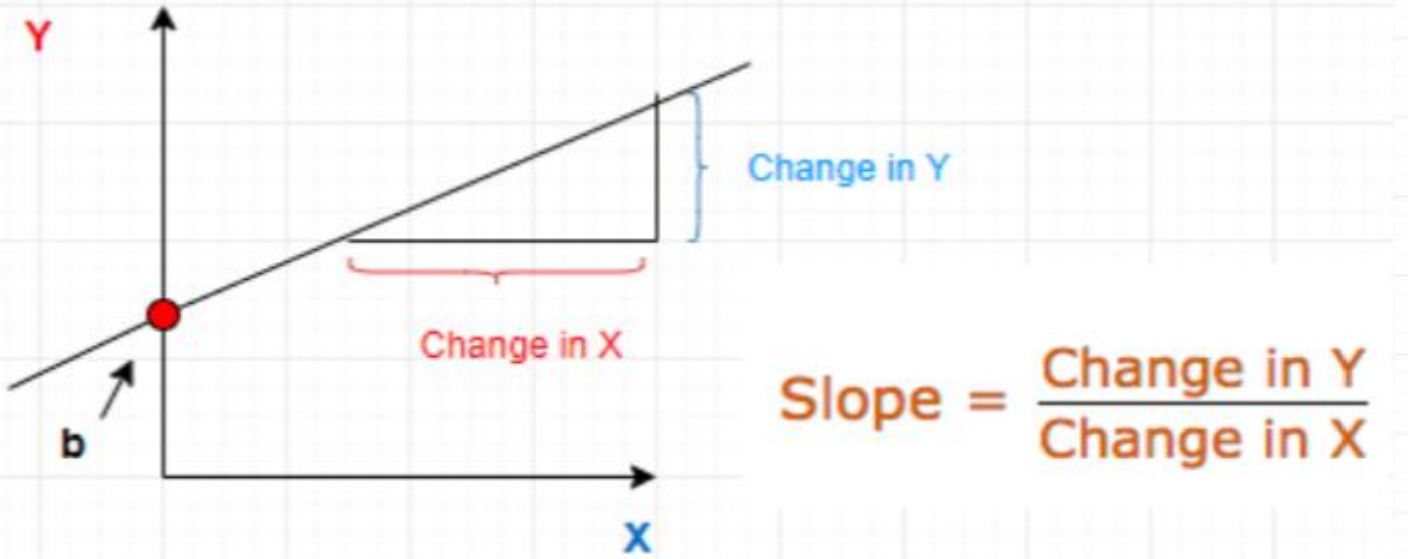
Machine Learning Model

- Consider a bunch of data points in a 2 D space. Assume that the data is related to the height and weight of a group of students.
- We are trying to predict some kind of relationship between these quantities so that we could predict the weight of some new students afterwards.

This is simple example of a supervised Machine Learning technique.

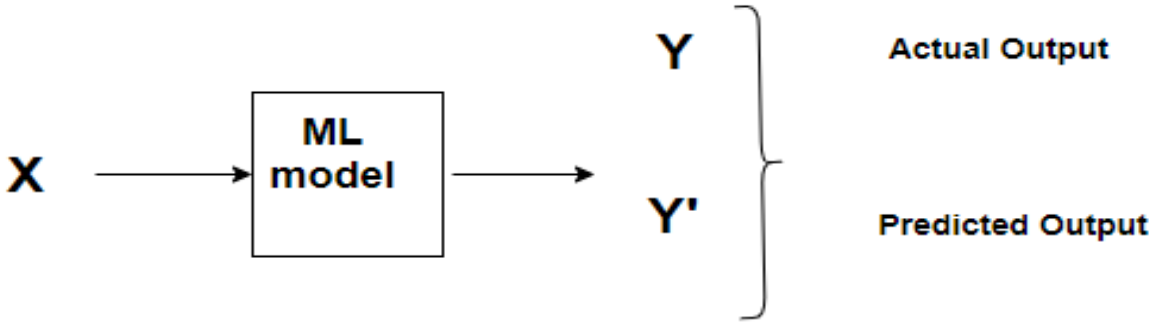
Exam: Let us now draw an arbitrary line in space that passes through some of these data points. The equation of this straight line would be Y = mX + b where m is the slope and b is its intercept on the Y-axis.





Predictions

- Given a known set of inputs and their corresponding outputs, A machine learning model tries to make some predictions for a new set of inputs.



ML process

The Error would be the difference between the two predictions.

Error = Y'(Predicted) - Y(Actual)

This relates to the a **Cost function or Loss function.**

Cost Function

- A **Cost Function/Loss Function** evaluates the performance of our Machine Learning Algorithm.
- The **Loss function** computes the error for a single training example while the **Cost functi**on is the average of the loss functions for all the training examples.

(A Cost function basically tells us ‘ how good’ our model is at making predictions for a given value of m and b.)

Lets ,there are a total of ’N’ points in the dataset and for all those ’N’ data points we want to minimize the error. So the Cost function would be the total squared error i.e

$$Cost = \frac{1}{N}\sum_{i=1}^{N}(Y' - Y)^2$$

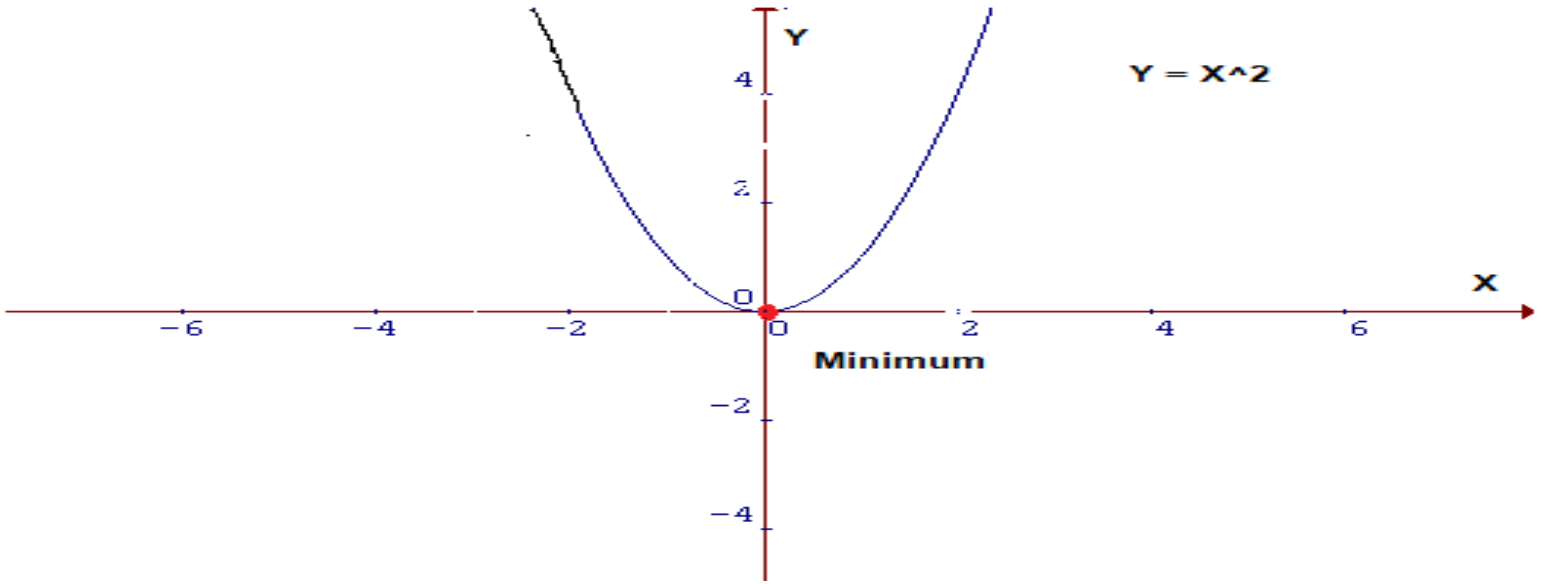
Why do we take the squared differences and simply not the absolute differences?
Because the squared differences make it easier to derive a regression line.
Indeed, to find that ***line we need to compute the first derivative of the Cost function,*** *and* ***it is much harder to compute the derivative of absolute values than squared values****. Also, the* ***squared differences increase the error distance****, thus, making the bad predictions more pronounced than the good ones.*

Minimizing the Cost Function

- The goal of any Machine Learning Algorithm is to minimize the Cost Function.
- This is because a lower error between the actual and the predicted values signifies that the algorithm has done a good job in learning. Since we want the lowest error value, we want those‘ **m’** and ‘**b’** values which give the smallest possible error.

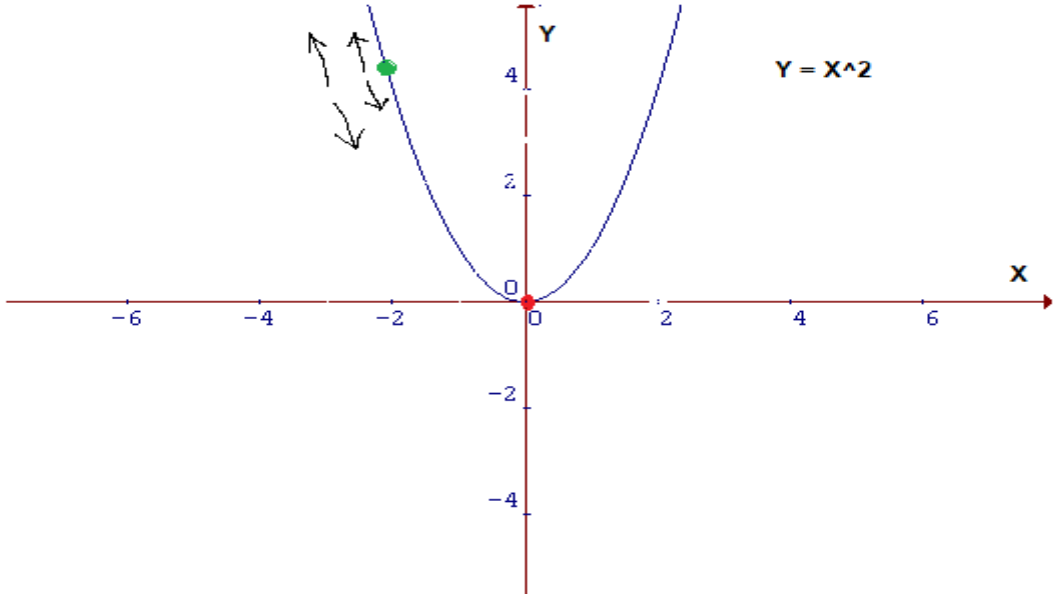
How do we actually minimize any function?

If we look carefully, our **Cost function** is of the form $Y = X^2$. In a Cartesian coordinate system, this is an equation for a parabola and can be graphically represented as :



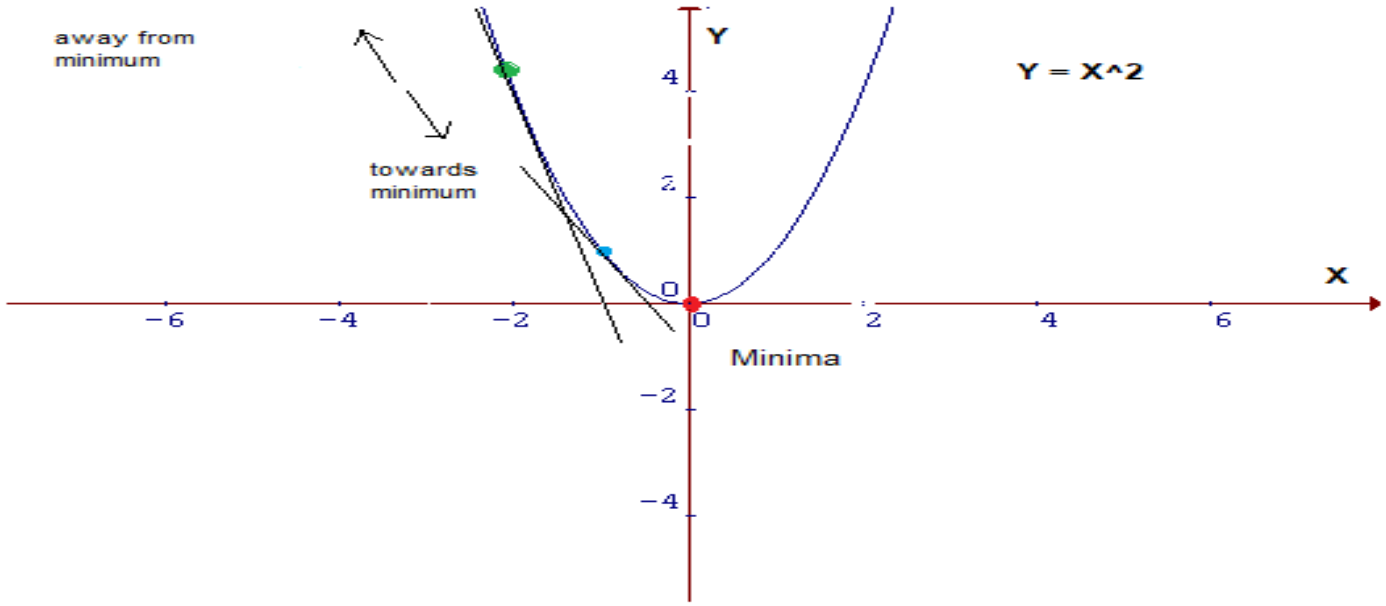
- To minimize the function above, we need to find **that value of X that produces the lowest value of Y.**
- which is the **red dot.** It is quite easy to locate the minima here since it is a 2D graph but this may not always be the case especially in case of higher dimensions. For those cases, we need to devise an algorithm to locate the minima, and that algorithm is called **Gradient Desce**
- **Gradient descent.**

.Consider that you are walking along the graph below, and you are currently at the '**green**' dot. Your aim is to reach the minimum i.e the 'red' dot, but from your position, you are unable to view it.



- **Possible actions would be:**
- You might go upward or downward
- If you decide on which way to go, you might take a bigger step or a little step to reach your destination.
- Gradient Descent Algorithm helps us to make these decisions efficiently and effectively with the use of derivatives.
- A **derivative** is a term that comes from calculus and is calculated as the slope of the graph at a particular point.
- The slope is described by drawing a tangent line to the graph at the point.
- So, if we are able to compute this tangent line, we might be able to compute the desired direction to reach the minima.

- **The Minimum Value**
- In the same figure, if we draw a tangent at the green point, we know that if we are moving upwards, we are moving away from the minima and vice versa. Also, the tangent gives us a sense of the steepness of the slope.



The slope at the blue point is less steep than that at the green point which means it will take much smaller **steps** to reach the minimum from the blue point than from the green point.

- **Mathematical Interpretation of Cost Function**

Mathematical Interpretation of Cost Function

Let us now put all these learnings into a mathematical formula. In the equation, `y = mX+b`, 'm' and 'b' are its parameters. During the training process, there will be a small change in their values. Let that small change be denoted by δ. The value of parameters will be updated as **m=m-δm** and **b=b-δb** respectively. Our aim here is to find those values of m and b in `y = mx+b` , for which the error is minimum i.e values which minimize the cost function.

Rewriting the cost function:

Parameters with small changes:

$$m = m - \delta m$$
$$b = b - \delta b$$

Given Cost Function for 'N' no of samples

$$Cost = \frac{1}{N}\sum_{i=1}^{N}(Y_i' - Y_i)^2$$

Cost function is denoted by J where J is a function of m and b

$$J_{m,b} = \frac{1}{N}\sum_{i=1}^{N}(Y_i' - Y_i)^2$$

Substituting the term Y'-Y with error for simplicity

$$J_{m,b} = \frac{1}{N}\sum_{i=1}^{N}(Error_i)^2$$

Auto-encoders

- Neural network to reconstruct the input

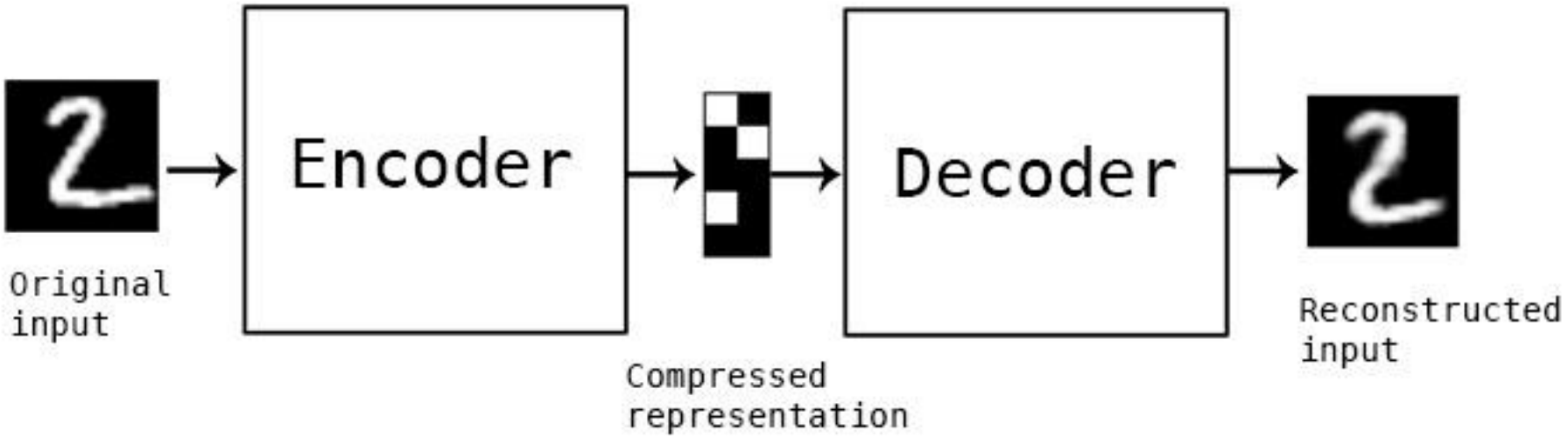
- An **auto encoder** is a type of artificial neural network used to learn efficient data coding in an unsupervised manner.

- The objective of an **auto encoder** is to learn a representation (encoding) for a set of data, typically for dimensionality reduction, by training the network.

Use of Auto encoder:

- Features reduction
- Unsupervised pre-training
- Problem of inter-subject variation

- Neural network to reconstruct the input
- Seems easy , no ??? Well, yes, but…
- Constraining the network yields interesting results



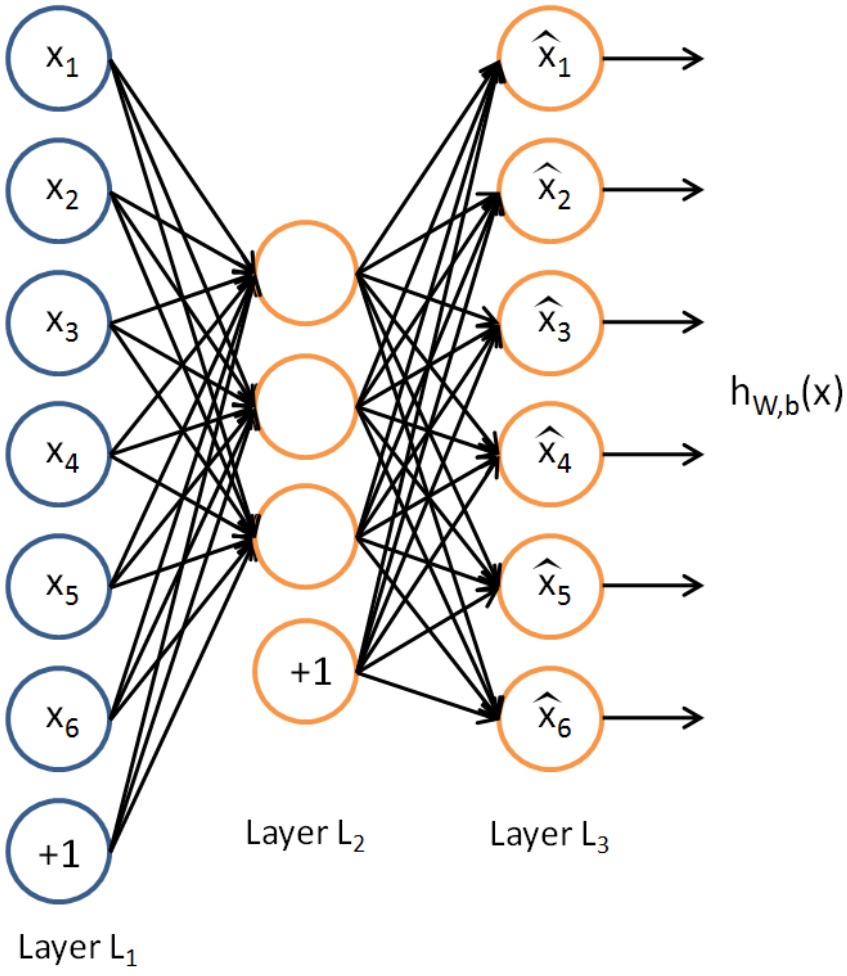
auto-encoders

- Suppose we have a set of unlabeled training examples

$$\{x_1, x_2, x_3, \ldots\}$$

- We want to train the network to output an estimation of the input

$$\hat{x} = h_{W,b} \approx x$$

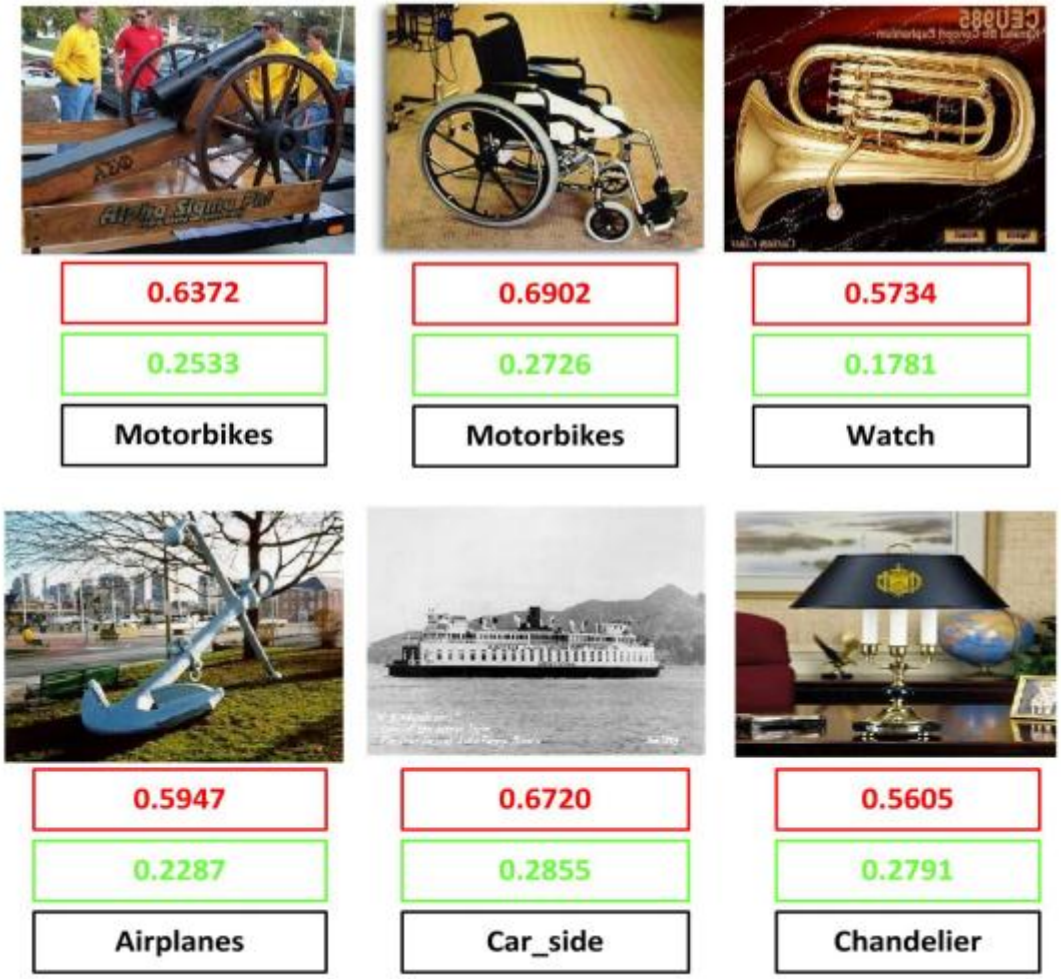


Auto-encoders – examples

- Web images search

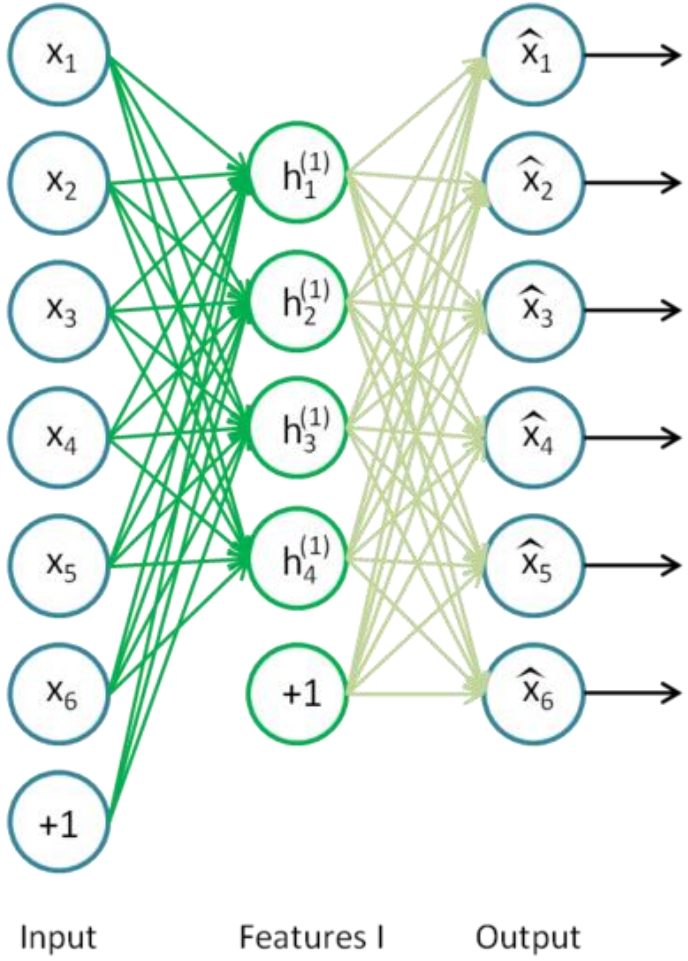
Auto-encoders – examples

- Anomaly detection
- Train on true images and find outliers by error.



Auto-encoders – how it works?

1. train a sparse autoencoder
   on the raw inputs x_k
   to find primary features $h_k{}^{(1)}$

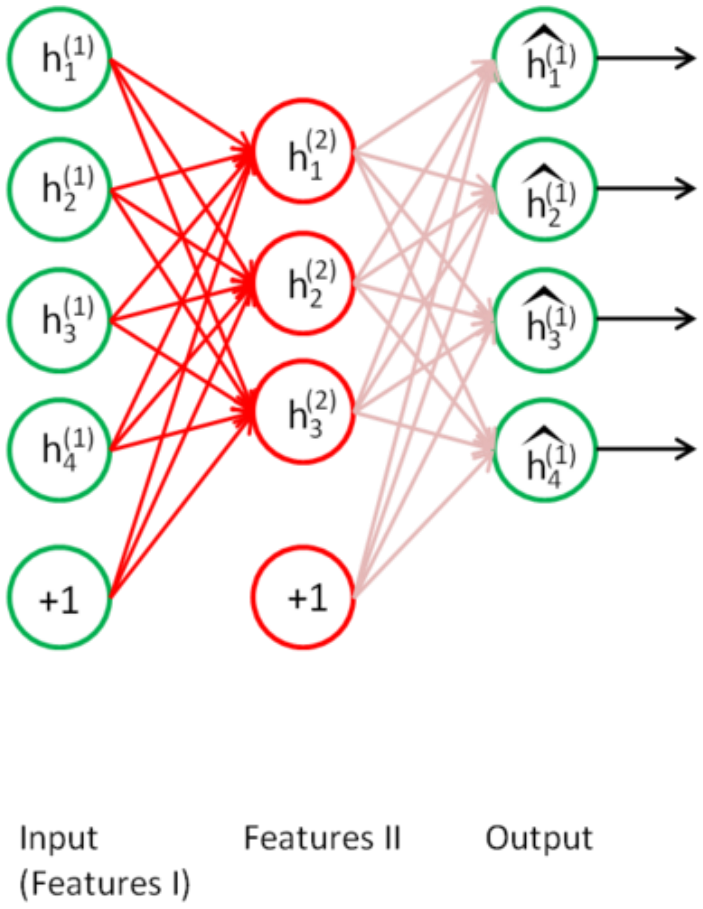


Auto-encoders – how it works?

2. use primary features as the "raw input" to another sparse autoencoder to learn secondary features

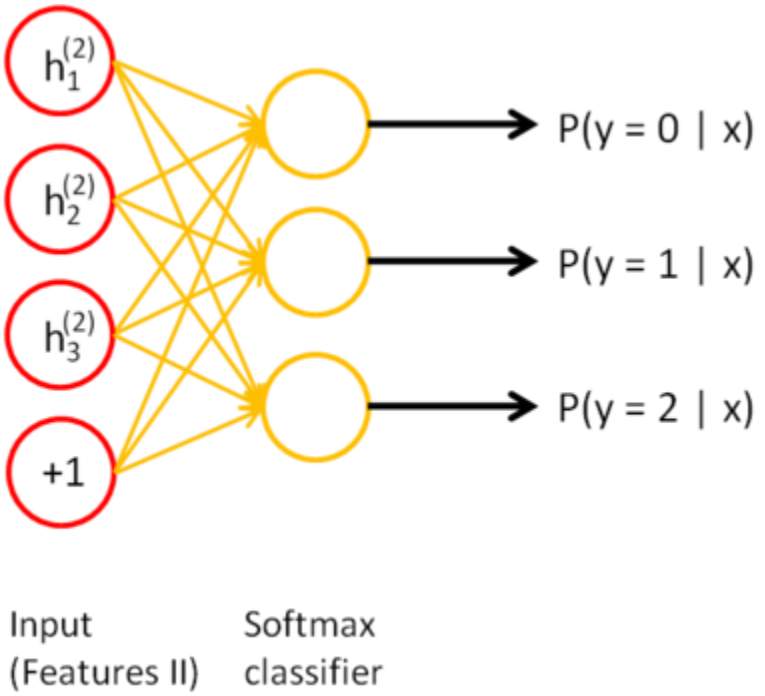
(next layer is smaller than the current layer)

2a. Repeat for N layers (if needed)



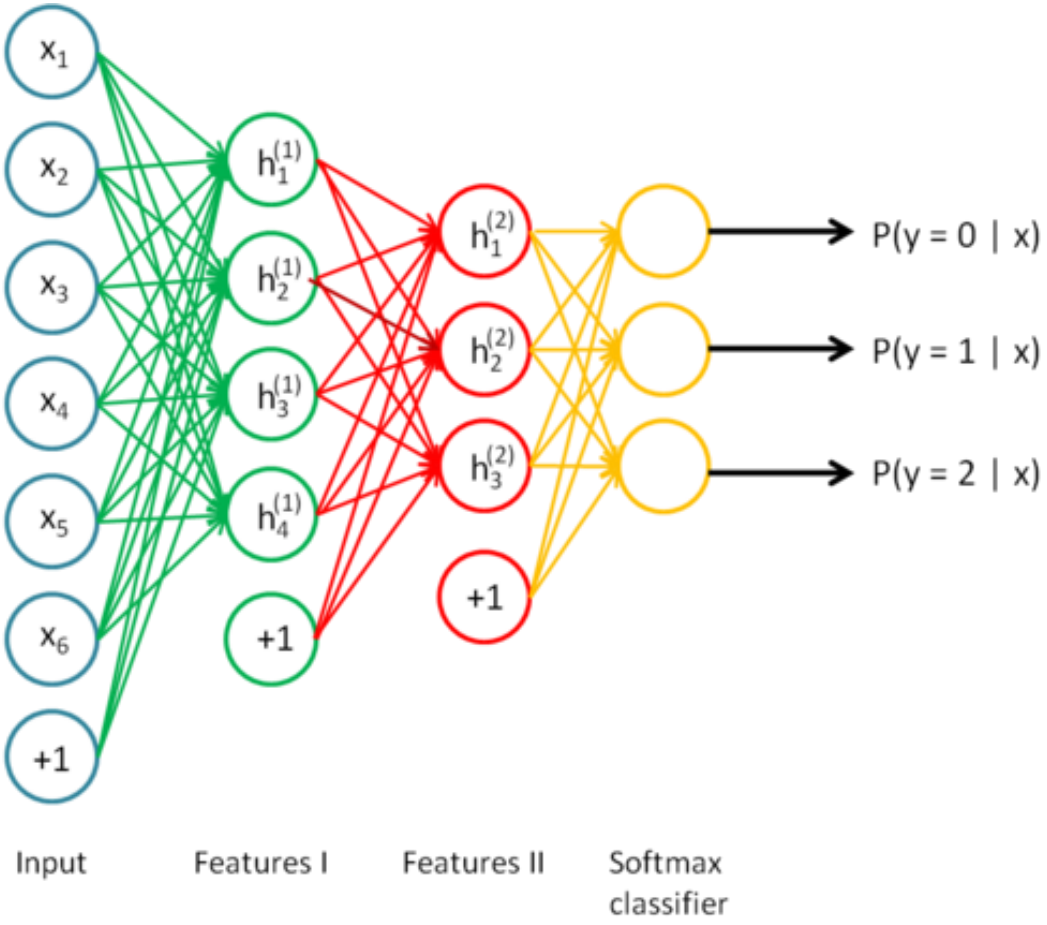
Auto-encoders – how it works?

3. Input last layer features into
   a softmax classifier



Auto-encoders – how it works?

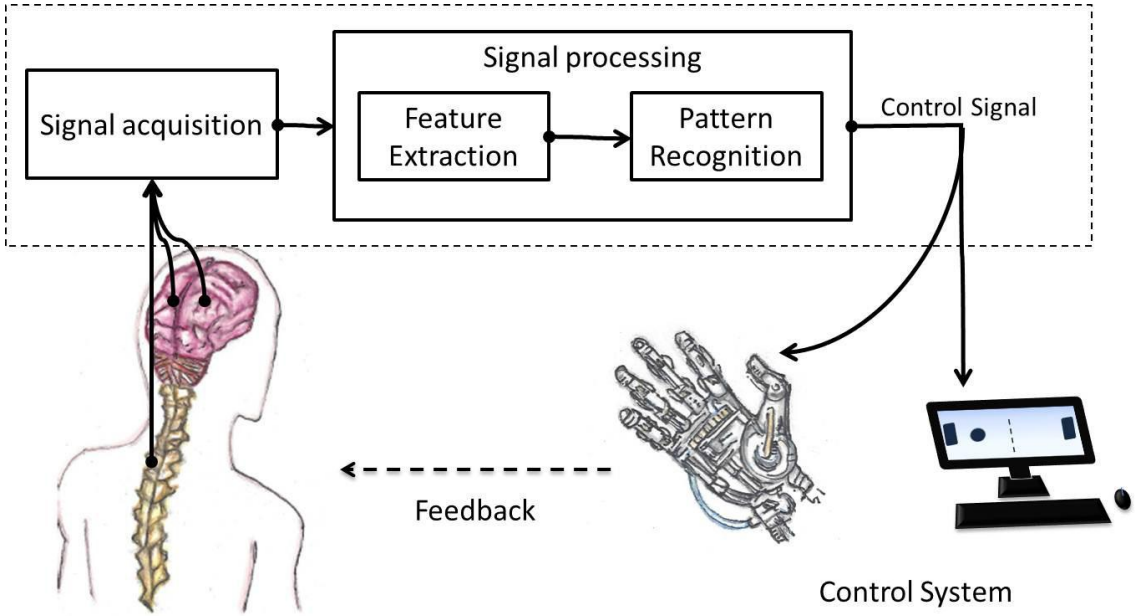
1. Combine all layers



Emotion Recognition

Emotion Recognition BCI

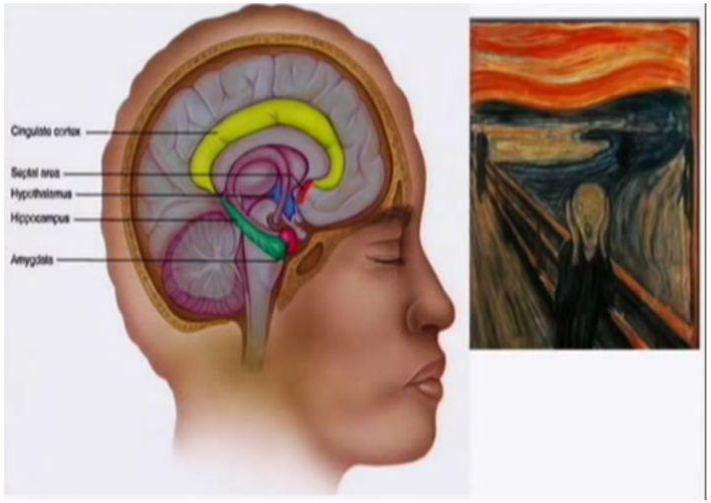
- Brain Computer Interface - BCI
- Better communication with computers for disable people.
- Detect patients' emotional responses to specific inexpressive faces.



Emotion Recognition

- Explain the motivation

Emotional brain



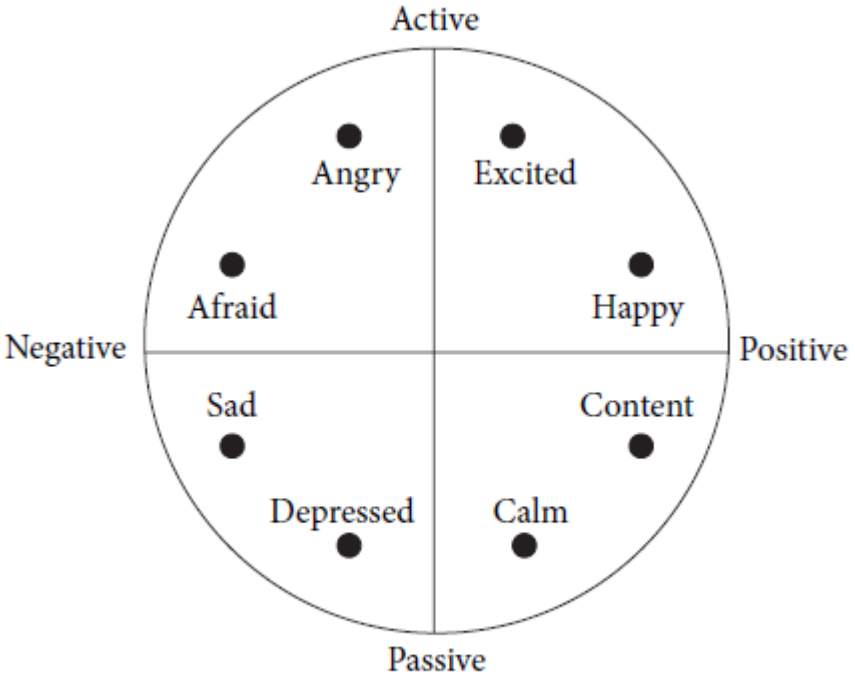
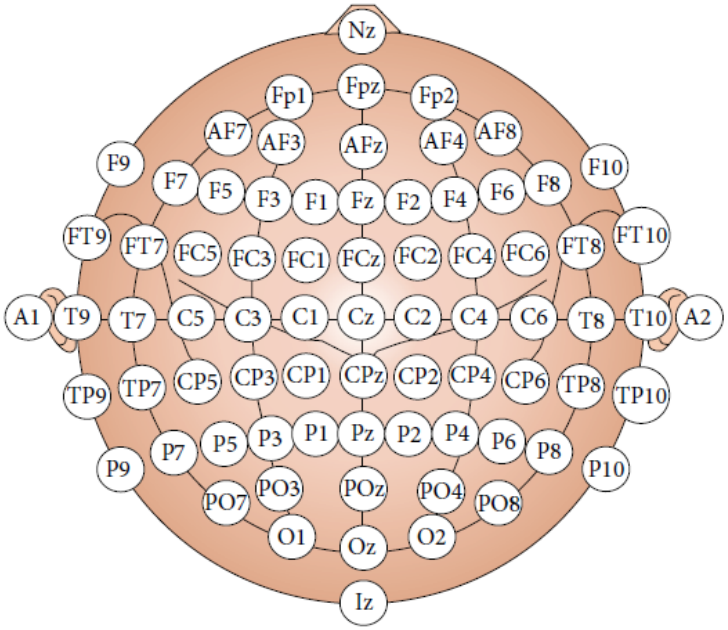
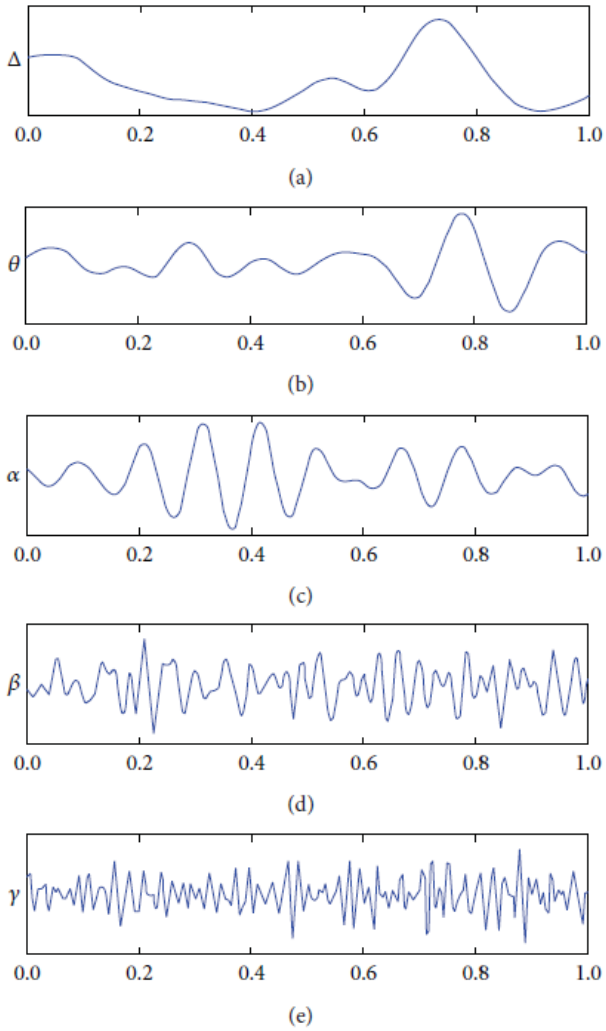


FIGURE 1: Valence-arousal dimensional model.

Raw data

- Time-Space-Frequency patterns
- Non-stationary
- Signals measured during specific task





Features extraction

Preprocessing

- Power spectral density (PSD)
- PSD divided into 5 frequency bands for 32 electrodes
- PSD difference between 14 symmetrical electrodes

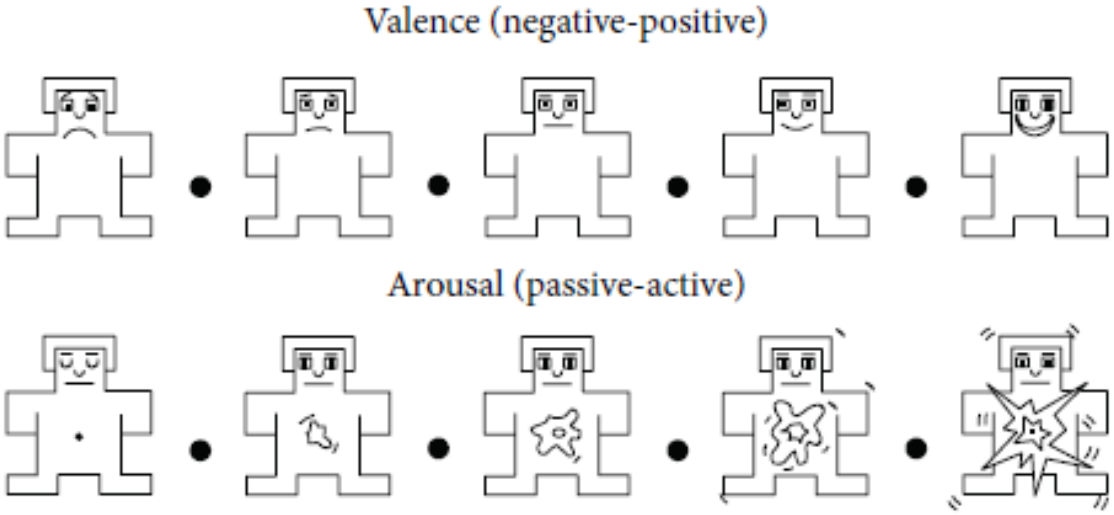
- Total of 230 input to the network

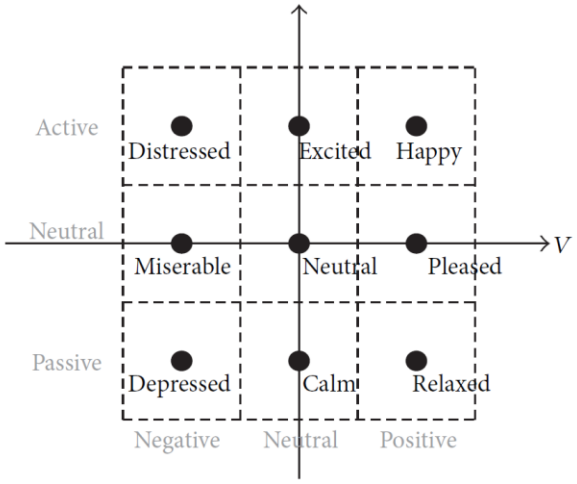
Normalization

- Baseline subtraction
- Re-scaling into [0.1,0.9] range

Data labeling

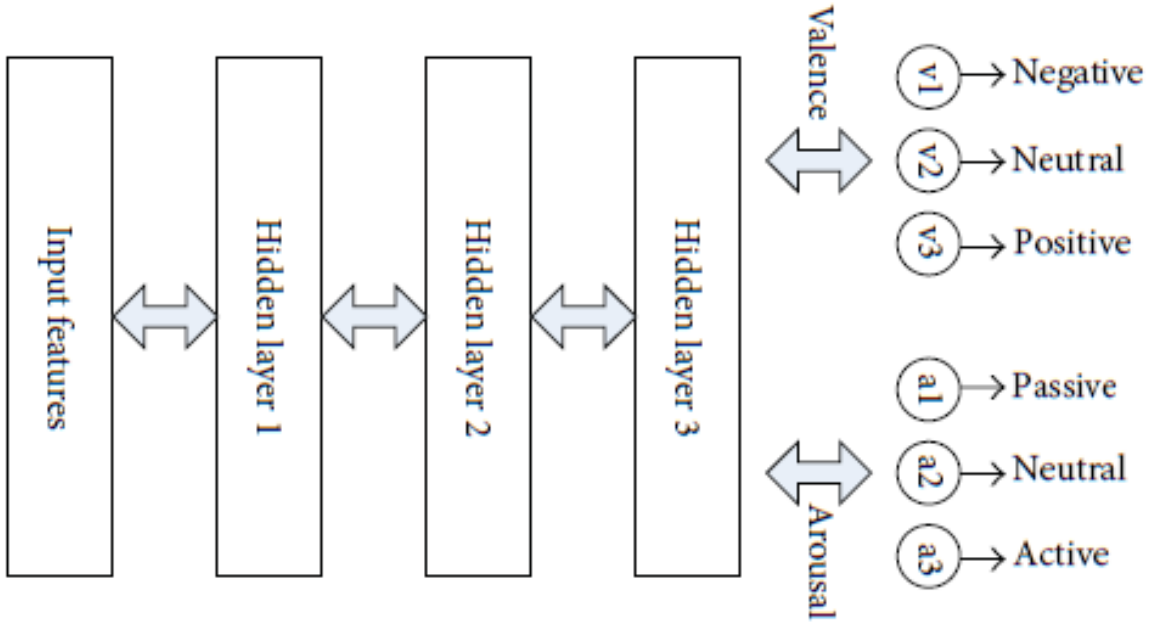
- 31 subjects
- DEAP
- 40 one-minute movies (per subject)
- After each movie – self assessment of arousal/valence level
- Total of 31 subject X 40 min movie X 60 PSDs/min = 74400 training sets

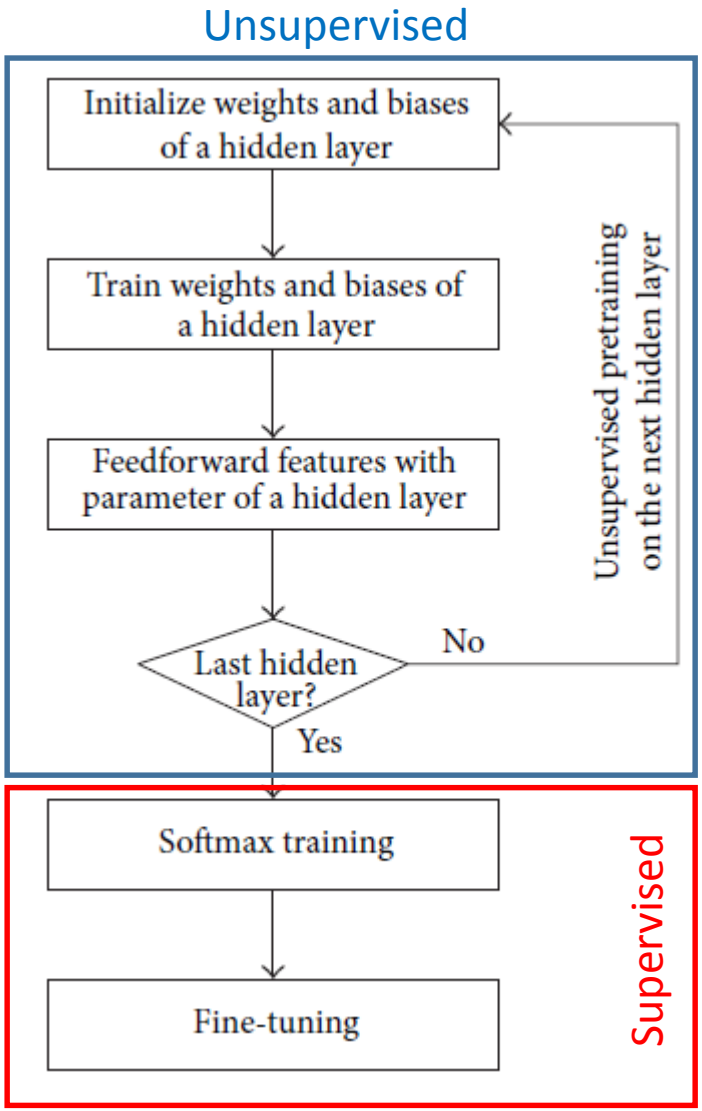




Network structure

- Sparce-autoencoder
- Three stacked hidden layers
- Two softmax classifiers
- Fine-tuning (back-prop.)





Autoencoder cost function

$$Cost = \frac{1}{2n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2$$

Autoencoder cost function

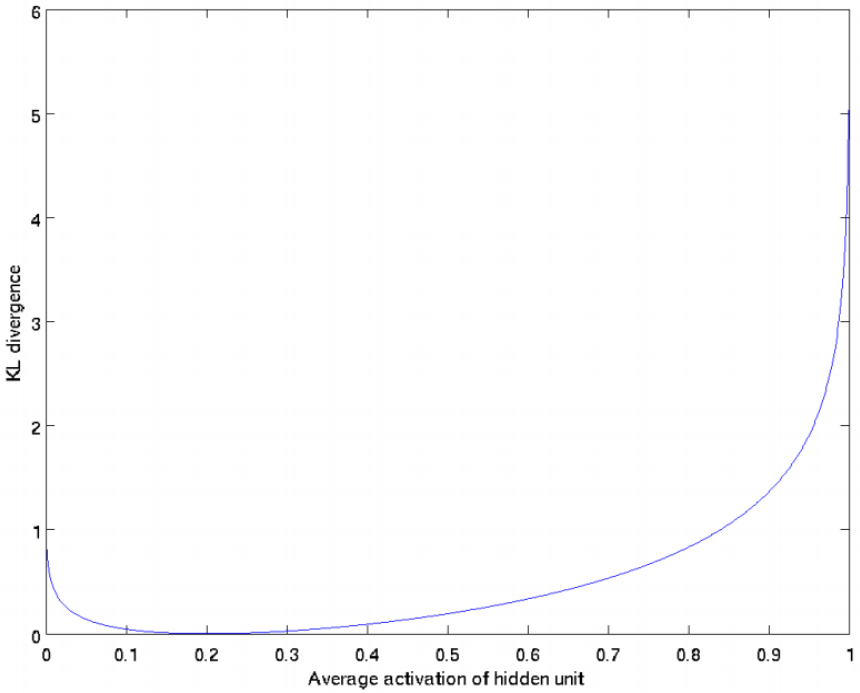
$$Cost = \frac{1}{2n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2 + \beta\sum_{j=1}^{m} KL(\rho \,\|\, \hat{\rho}_j)$$

$$KL(\rho||\hat{\rho}_j) = \rho\log\frac{\rho}{\hat{\rho}_j} + (1-\rho)\log\frac{1-\rho}{1-\hat{\rho}_j}$$

Kullback-Leibler (KL) divergence.

Standard function for measuring difference between two Bernoulli distributions.

If $\rho = \hat{\rho}_j$ KL=0



Autoencoder cost function

$$Cost = \frac{1}{2n}\sum_{i=1}^{n}(x_i - \hat{x}_i)^2 + \beta\sum_{j=1}^{m} KL(\rho \,\|\, \hat{\rho}_j) + \frac{\lambda}{2}\sum_{i=1}^{n}\sum_{j=1}^{m} W_{ij}^{\,2}$$

- λ is called the weight decay parameter
- Used to limit the network weights.

L1 and L2 Regularization

- Regularization is a process of introducing additional information in order to prevent overfitting loss function.
- A regression model that uses **L1 regularization** technique is called Lasso Regression and model which uses **L2** is called Ridge Regression. The key difference between these two is the penalty term.
- Ridge regression adds “squared magnitude” of coefficient as penalty.
- Gradient descent is simply a method to find the ‘right’ coefficients through iterative updates using the value of the gradient.

(gradient descent can be used in a simple linear regression)

L1 and L2 Regularization

Regularization is a process of introducing additional information in order to prevent **overfitting** loss function.

A regression model that uses **L1 regularization** technique is called Lasso Regression and model which uses **L2** is called Ridge Regression. The key difference between these two is the penalty term.

Ridge regression adds “squared magnitude” of coefficient as penalty.

Gradient descent is simply a method to find the ‘right’ coefficients

through iterative updates using the value of the gradient.

(gradient descent can be used in a simple linear regression)

L2 Regularization:

A regression model that uses L1 regularization technique is called ***Lasso Regression*** and model which uses L2 is called ***Ridge Regression***.

- **Ridge regression** adds "*squared magnitude*" of coefficient as penalty term to the loss function.

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

Cost function

- If *lambda* is zero then you can imagine we get back OLS(ordinary Least Square). However, if *lambda* is very large then it will add too much weight and it will lead to under-fitting. Having said that it's important how *lambda* is chosen.
- This technique to avoid over-fitting issue.
- **Lasso Regression** (Least Absolute Shrinkage and Selection Operator) adds "*absolute value of magnitude*" of coefficient as penalty term to the loss function.

$$\sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

Cost function

- if *lambda* is zero then we will get back OLS whereas very large value will make coefficients zero hence it will under-fit.

- The **key difference** between these techniques is that Lasso shrinks the less important feature’s coefficient to zero thus, removing some feature altogether. So, this works well for **feature selection** in case we have a huge number of features.
- Traditional methods like cross-validation, stepwise regression to handle overfitting and perform feature selection work well with a small set of features **but** these techniques are a great alternative when we are dealing with a large set of features.

Batch Normalization

- We normalize the input layer by adjusting and scaling the activations. For example, when we have features from 0 to 1 and some from 1 to 1000, we should normalize them to speed up learning.

How does batch normalization work?

To increase the stability of a neural network, batch normalization normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

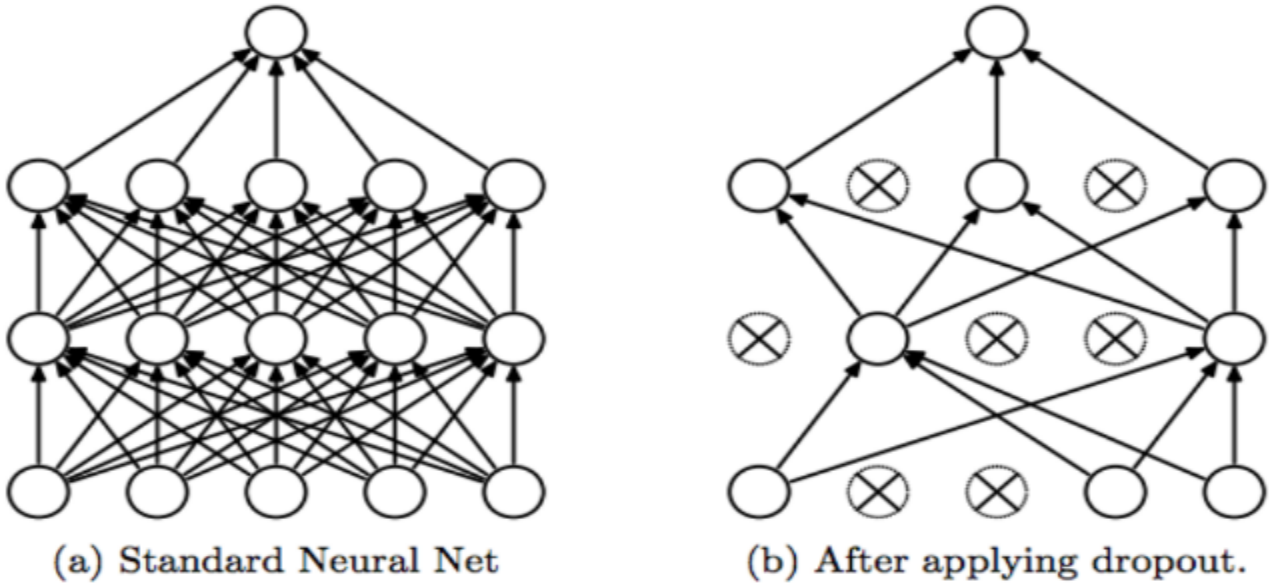
$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

- Dropout Neural Network Layer
- Dropout is a technique used to **prevent a model from overfitting**. Dropout works by randomly setting the outgoing edges of hidden units (neurons that make up hidden layers) to 0 at each update of the training phase.



(a) Standard Neural Net (b) After applying dropout.

Dropout Neural Network Layer In

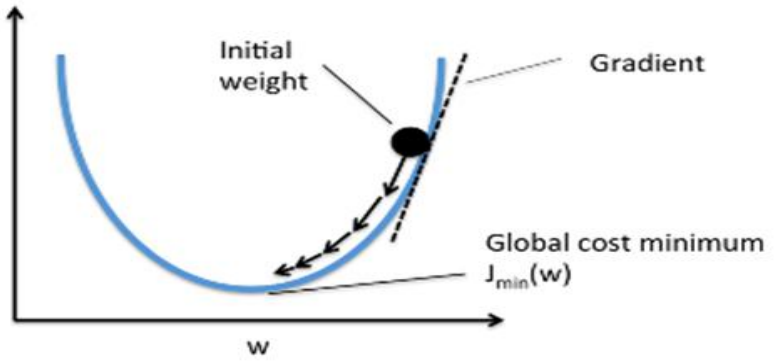
hyper-parameters:

L**earning rate**, **batch size**, **momentum**, and **weight decay**.

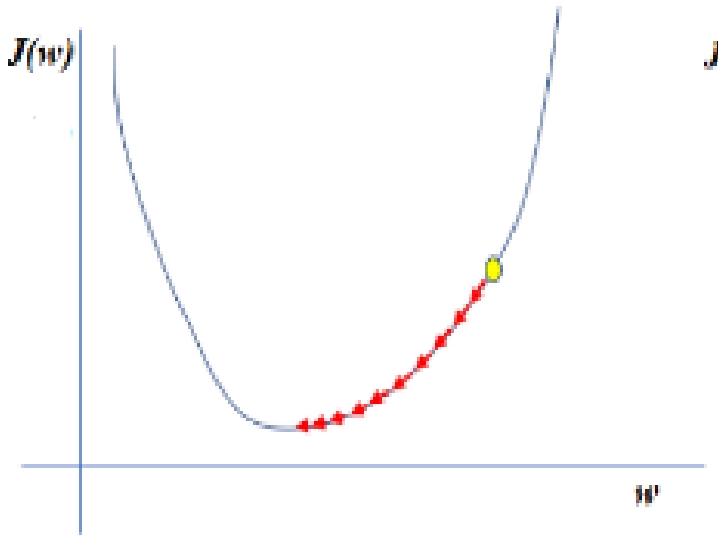
These hyper-parameters act as holds which can be twisted during the training of the model. For our model to provide best result, we need to find the optimal value of these **hyper-Parameter.**

Gradient Descent

Gradient descent is an optimization technique commonly used in training machine learning algorithms. The main aim of training ML algorithms is to adjust the weights w to minimize the loss or cost. This cost is measure of how well our model is doing, we represent this cost by J(w) Thus, by minimizing the cost function we can find the optimal parameters.

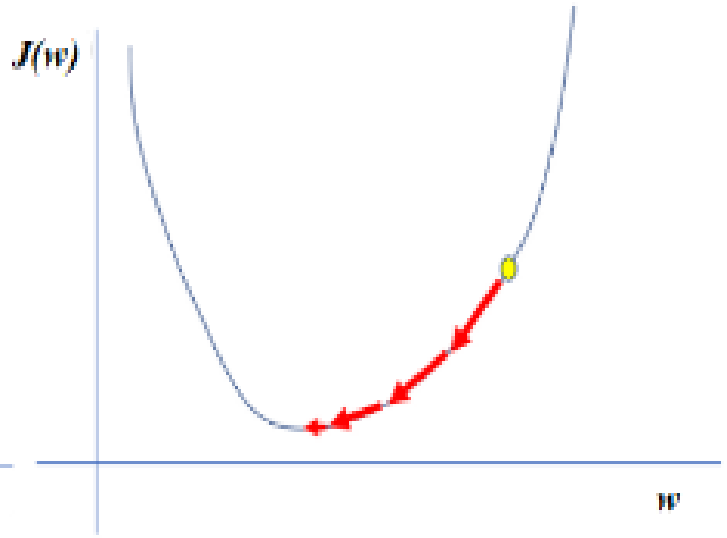


Too low



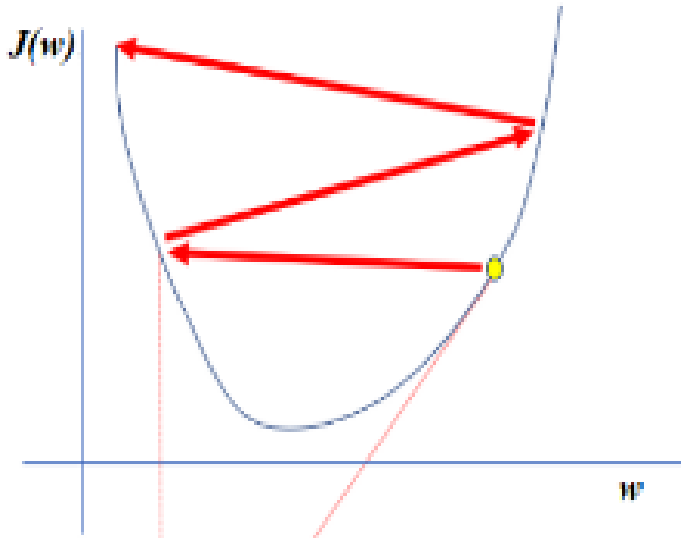
A small learning rate requires many updates before reaching the minimum point

Just right



The optimal learning rate swiftly reaches the minimum point

Too high



Too large of a learning rate causes drastic updates which lead to divergent behaviors

CNN (convolutional Neural Network)

- Convolutional layers are the layers where filters are applied to the original image, or to other feature maps in a deep CNN.
- The name “convolutional neural network” indicates that the network employs a mathematical operation called convolution .
- **Convolution** is a specialized kind of linear operation.
- Convolutional networks are simply neural networks that use convolution in place of general **matrix multiplication** in **at least one** of their layers
- Convolutional kernels defined by a **width and height** (hyper-parameters).

What if we learned the features to detect?

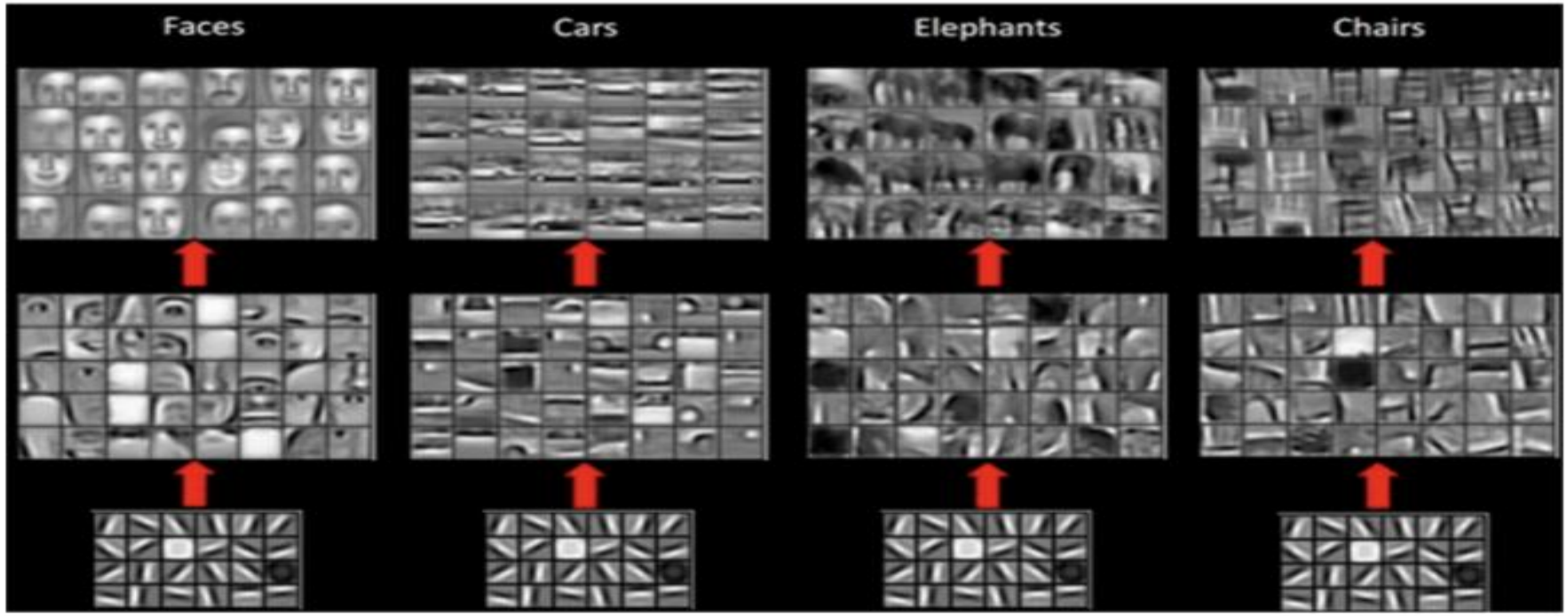
- We need a system that can do Representation Learning (or Feature Learning).
- Representation Learning is a technique that allows a system to automatically find relevant features for a given task. Replaces manual feature engineering. There are several techniques for this:

(1)Unsupervised (K-means, PCA, …)

(2)Supervised (Sup. Dictionary learning, Neural Networks!)

Working of CNN:

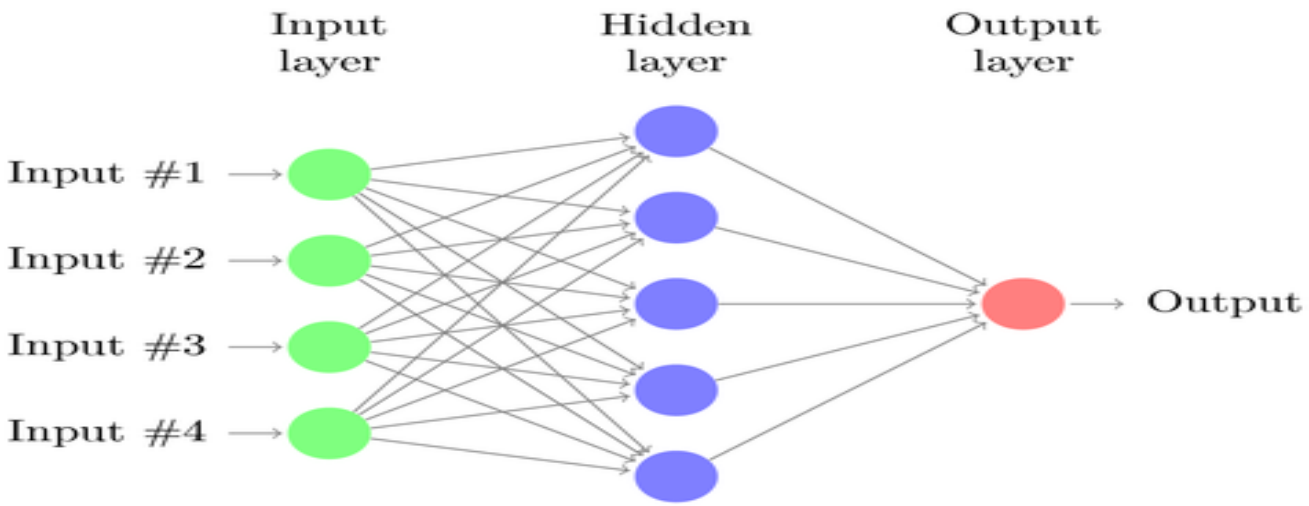
- Each CNN layer learns filters of increasing complexity.
- The first layers learn basic feature detection filters: edges, corners, etc
- The middle layers learn filters that detect parts of objects. For faces, they might learn to respond to eyes, noses, etc
- The last layers have higher representations: they learn to recognize full objects, in different shapes and positions. D example of a CNN working in practice



Examples of CNN's trained to recognize specific objects and their generated feature maps.

The Problem with Traditional Neural Networks or MLP

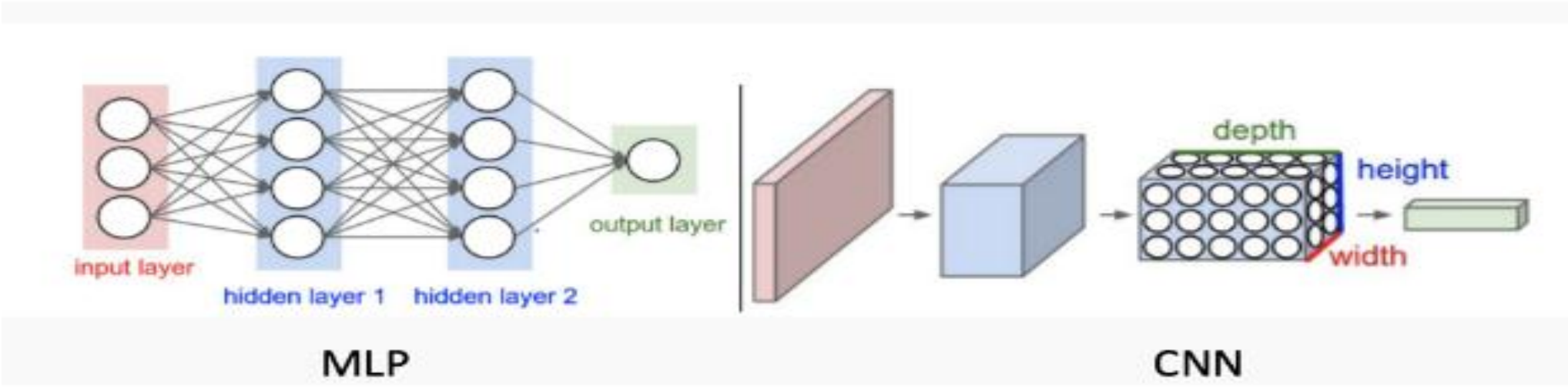
- Traditional neural networks called the multilayer perceptron (MLP).
- If you are not familiar with these, there are hundreds of tutorials on Medium outlining how MLPs work.
- These are modeled on the human brain, whereby neurons are stimulated by connected nodes and are only activated when a certain threshold value is reached.
- **(problem) :** Do not scale well for images
- Ignore the information brought by pixel position and correlation with neighbors
- Cannot handle translations.



A standard multilayer perceptron (traditional neural network).

Comparison of architecture for MLP and CNN:

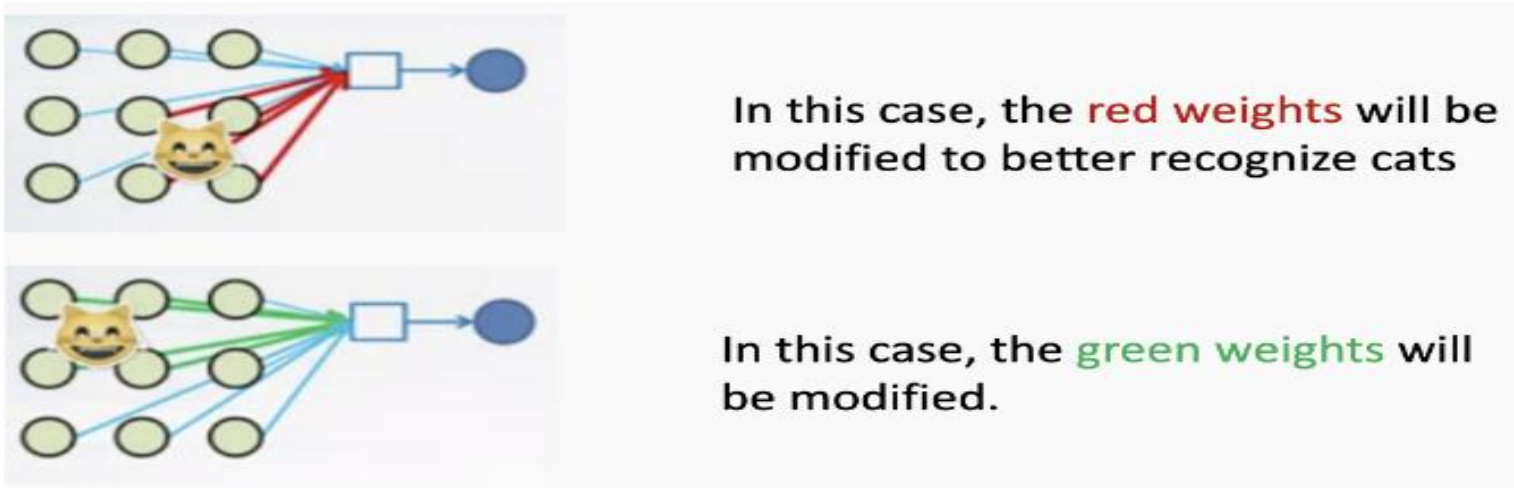
- CNN's are also composed of layers, but those layers are not fully connected.
- They have filters, sets of cube-shaped weights that are applied throughout the image.
- Each 2D slice of the filters are called kernels. These filters introduce translation invariance and parameter sharing.



Comparison of architecture for MLP and CNN.

There are several drawbacks of MLP's

- especially when it comes to image processing. MLPs use one perceptron for each input (e.g. pixel in an image, multiplied by 3 in RGB case).
- The amount of weights rapidly becomes unmanageable for large images. For a 224 x 224 pixel image with 3 color channels there are around 150,000 weights that must be trained! As a result, difficulties arise whilst training and overfitting can occur.
- Clearly, MLPs are not the best idea to use for image processing. One of the main problems is that spatial information is lost.



A cat detector using an MLP which changes as the position of the cat changes.

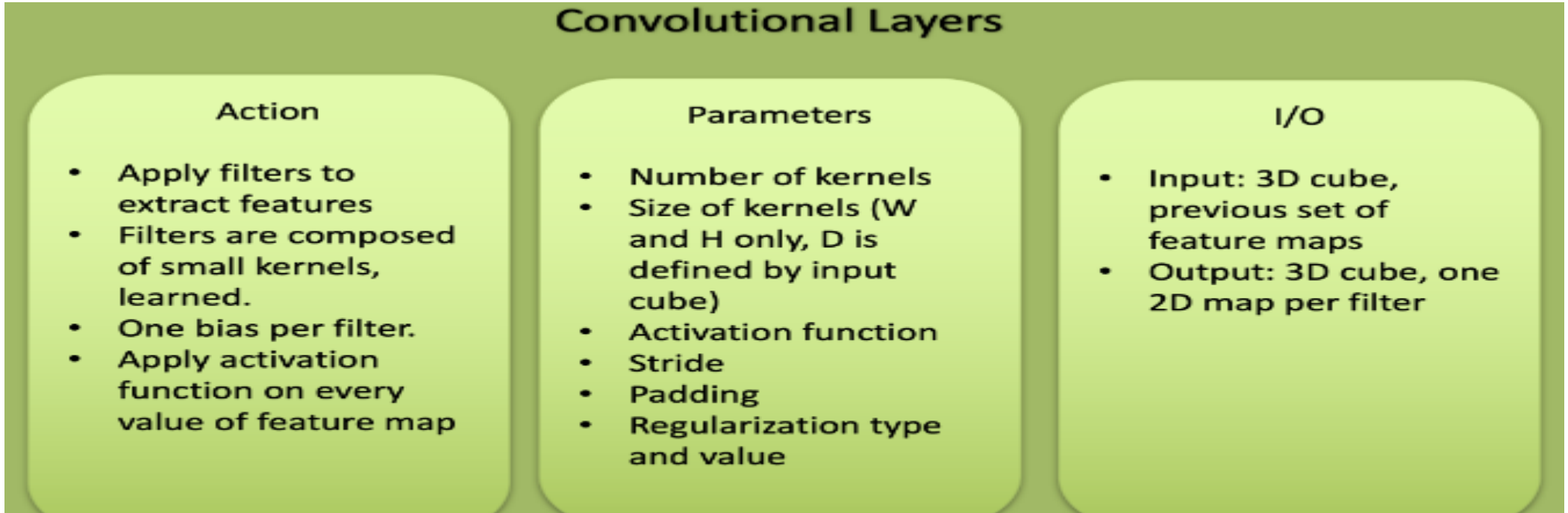
Types of Layers in a convolutional neural network**:**

- There are three types of layers in a convolutional neural network.

(1) convolutional layer(2)pooling layer (3)fully connected layer

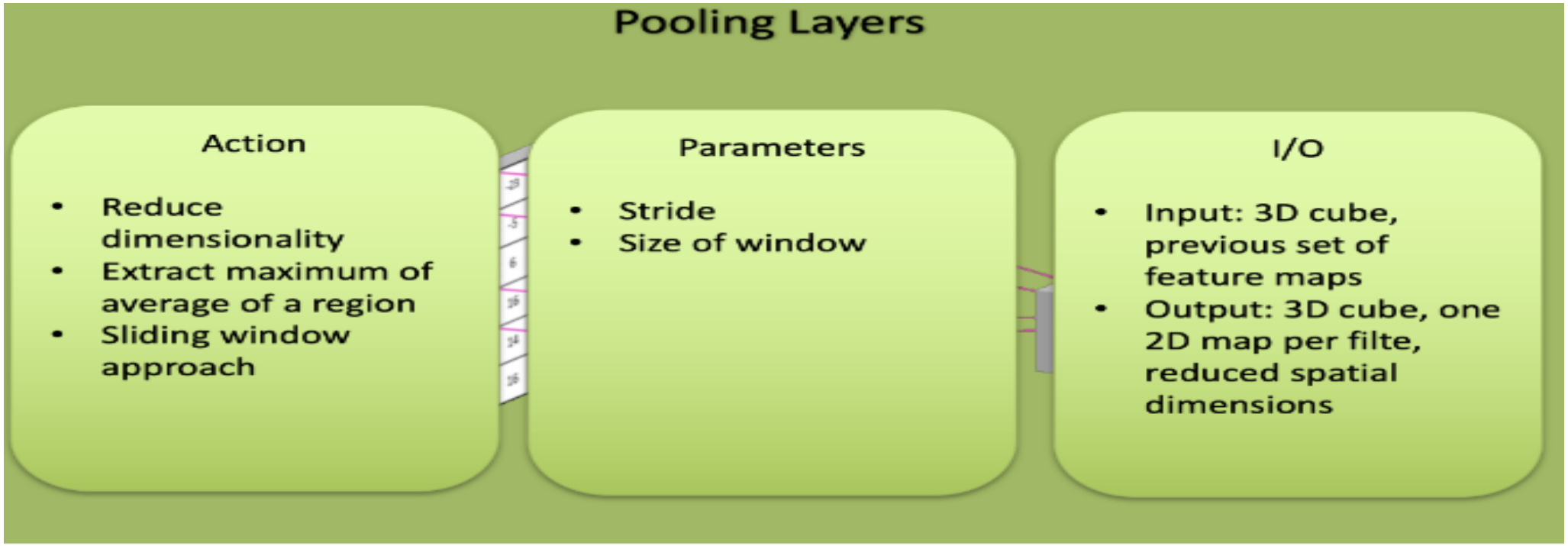
(1)convolutional layer:

convolutional layer Convolutional layers are the layers where filters are applied to the original image, or to other feature maps in a deep CNN. This is where most of the user-specified parameters are in the network. The most important parameters are the number of kernels and the size of the kernels.



Pooling layers:

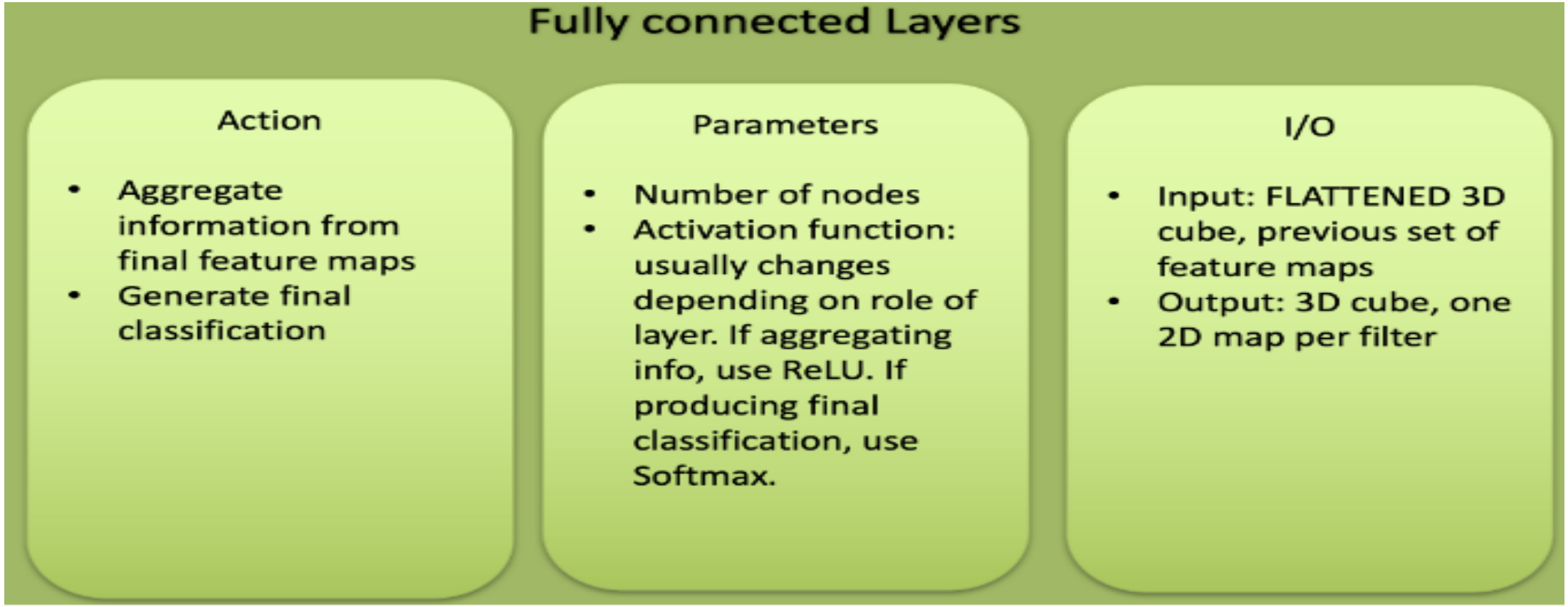
- Pooling layer are similar to convolutional layers, but they perform a specific function such as max pooling, which takes the maximum value in a certain filter region, or average pooling, which takes the average value in a filter region.
- These are typically used to reduce the dimensionality of the network.



Features of a pooling layer

Connected Layers:

Fully connected layers are placed before the classification output of a CNN and are used to flatten the results before classification. This is similar to the output layer of an MLP.



Features of a fully connected layer.

Use of CNN:

The general idea of CNN's is to intelligently adapt to the properties of images:

- Pixel position and neighborhood
- Elements of interest can appear anywhere in the image

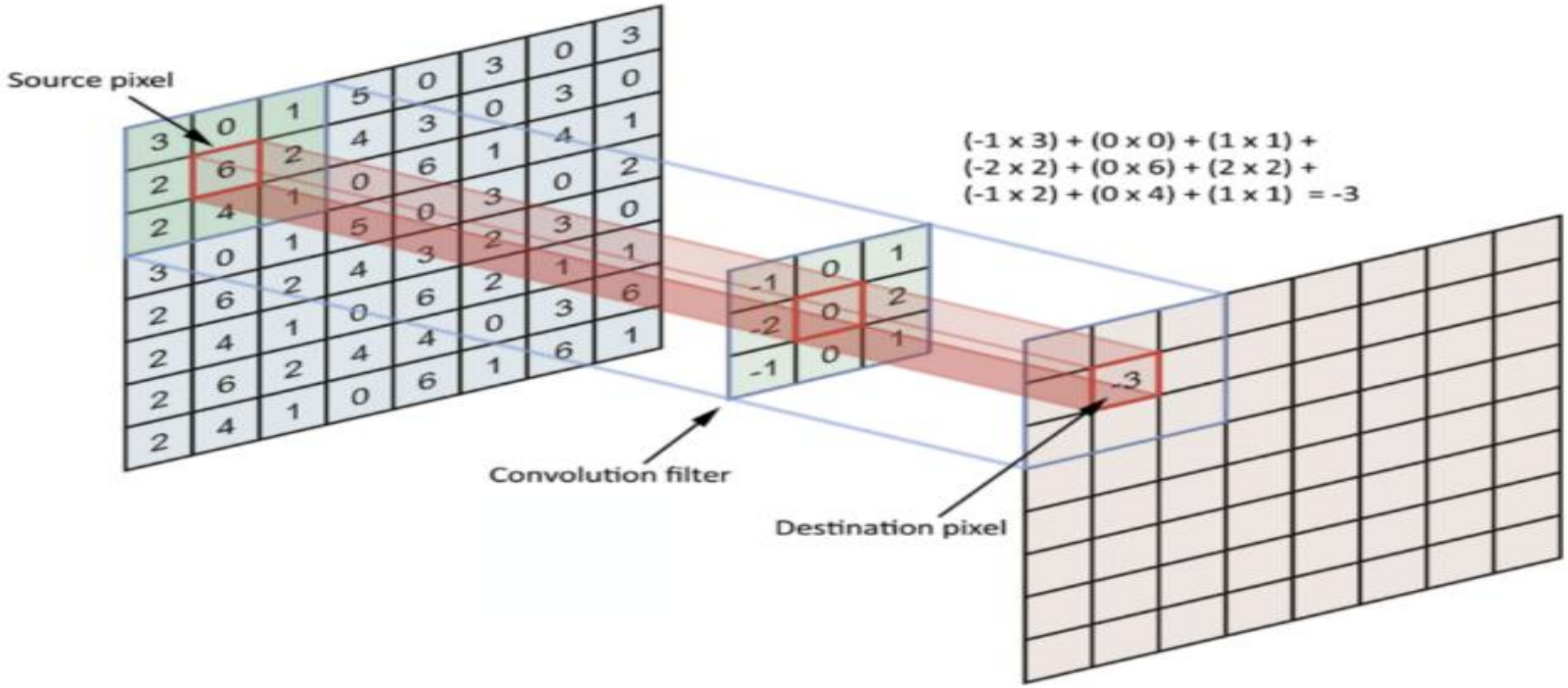
The convolution operation:

We analyze the influence of nearby pixels by using something called a filter.

A filter is exactly what you think it is, in our situation, we take a filter of a size specified by the user (a rule of thumb is 3x3 or 5x5) and we move this across the image from top left to bottom right.

For each point on the image, a value is calculated based on the filter using a convolution operation.

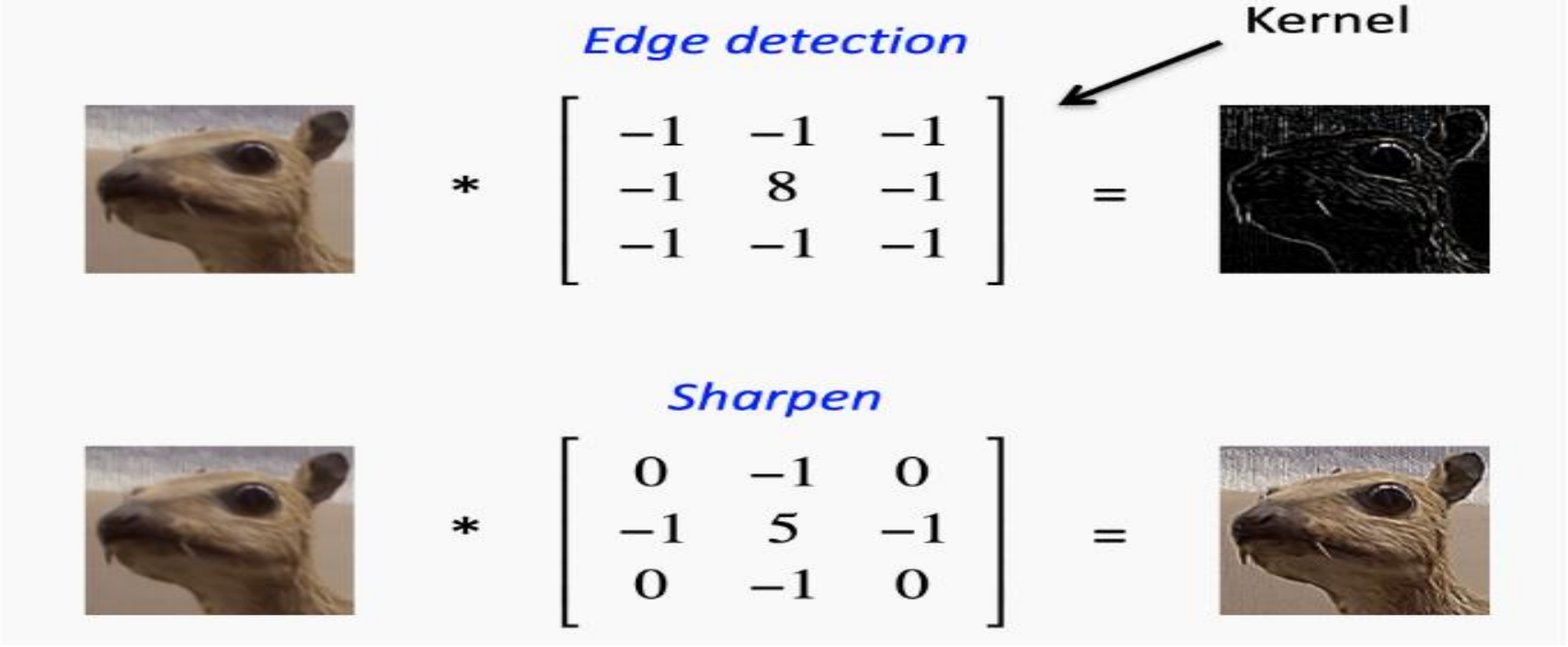
A filter could be related to anything, for pictures of humans, one filter could be associated with seeing noses, and our nose filter would give us an indication of how strongly a nose seems to appear in our image, and how many times and in what locations they occur.



The convolution operation.

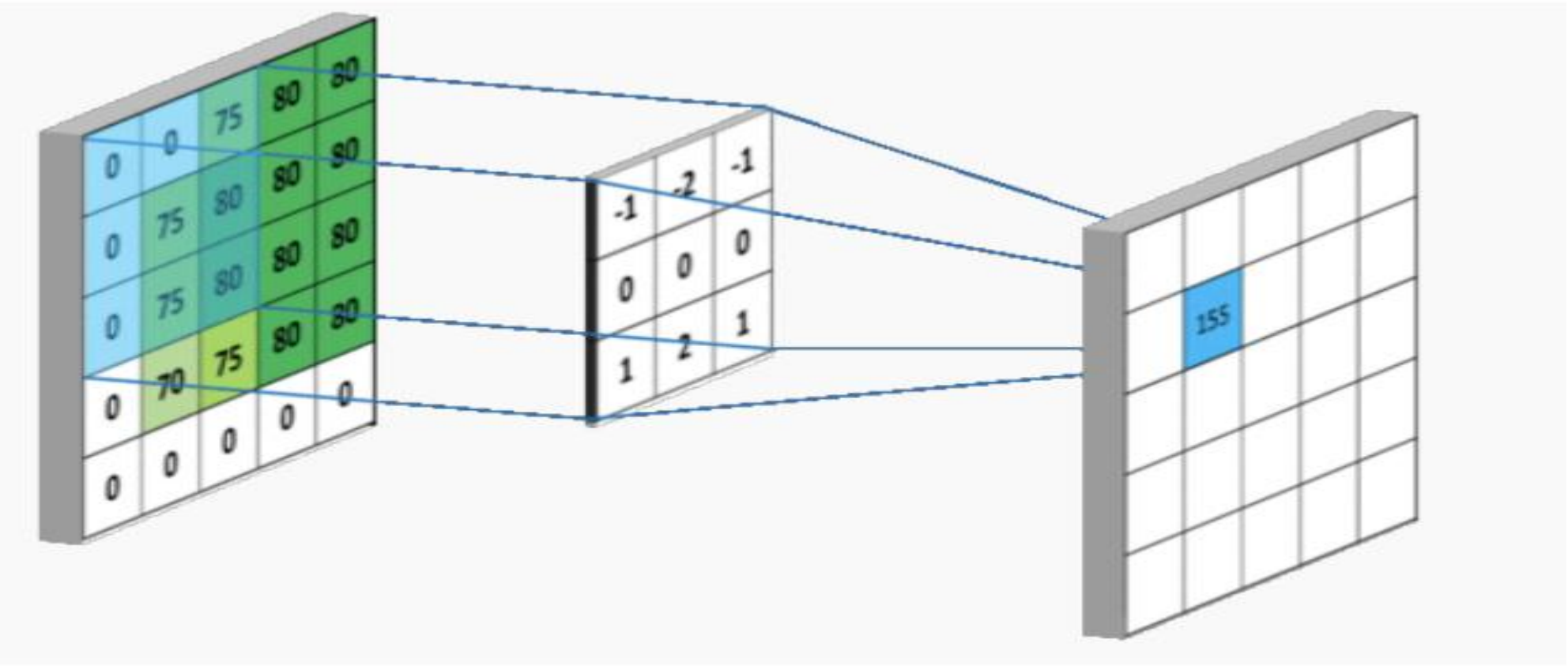
Examples of filters or kernels :

After the filters have passed over the image, a feature map is generated for each filter. These are then taken through an activation function, which decides whether a certain feature is present at a given location in the image. We can then do a lot of things, such as adding more filtering layers and creating more feature maps, which become more and more abstract as we create a deeper CNN.



Examples of kernel filters for CNN's.

Example of how convolutions are applied to images using kernel filters:



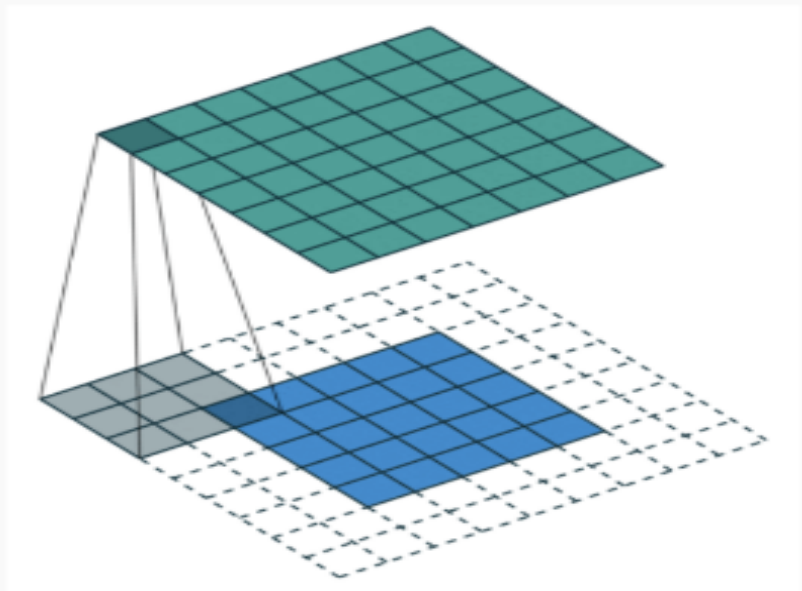
Padding:

Padding essentially makes the feature maps produced by the filter kernels the same size as the original image.

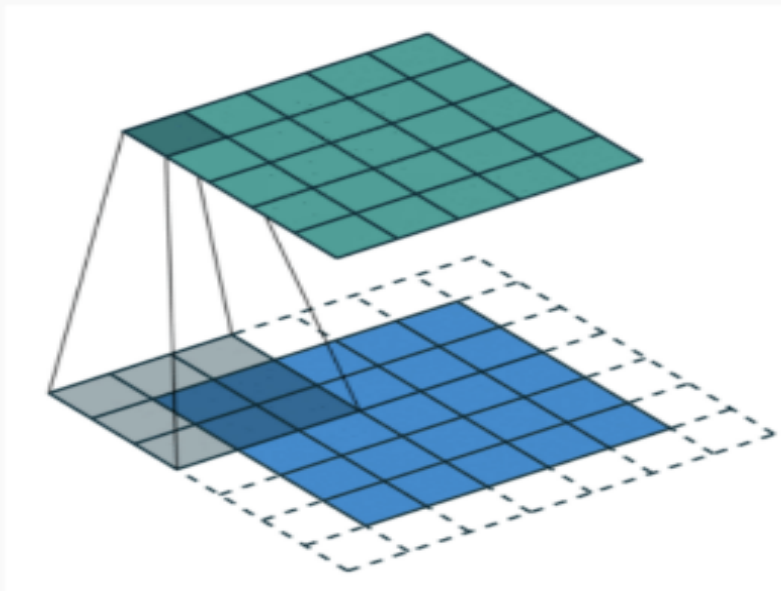
This is very useful for deep CNN's as we don't want the output to be reduced so that we only have a 2x2 region left at the end of the network upon which to predict our result.

Why we Need of Padding?

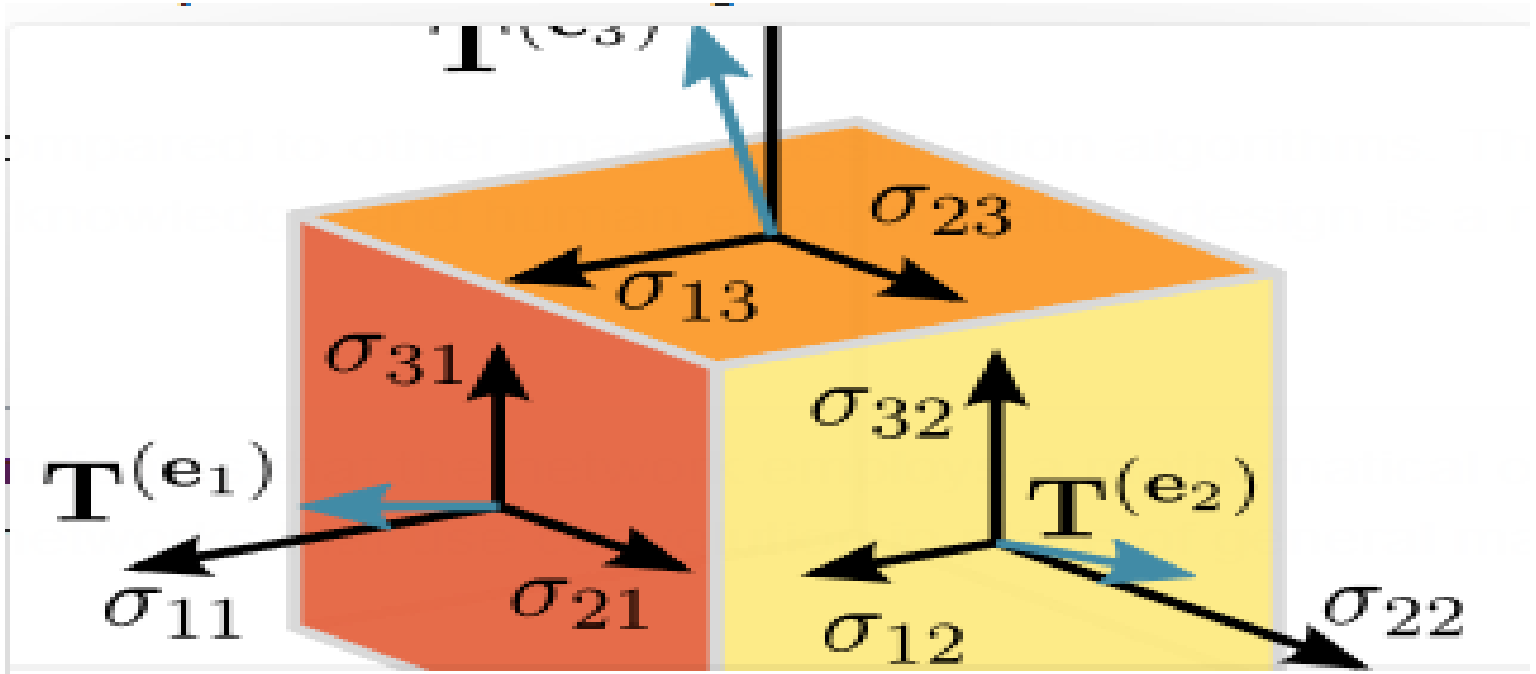
- If we apply convolutions on a normal image, the result will be down-sampled by an amount depending on the size of the filter.
- What do we do if we don't want this to happen? We can use padding.



Full padding. Introduces zeros such that all pixels are visited the same amount of times by the filter. Increases size of output.



Same padding. Ensures that the output has the same size as the input.



In mathematics, a **tensor** is an algebraic object that describes a (multilinear) relationship between sets of algebraic objects related to a vector space. Objects that tensors may map between include vectors and scalars, and, recursively, even other tensors. Tensors can take several different forms

1x1 convolution :

simply maps an input pixel with all it's channels to an output pixel, not looking at anything around itself. It is often used to reduce the number of depth channels, since it is often very slow to multiply volumes with extremely large depths.

Inception Network: most popular CNNs just stacked convolution layers deeper and deeper, hoping to get better performance

Recurrent neural network:

A **recurrent neural network** (**RNN**) is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behavior. Derived from feedforward neural networks, RNNs can use their internal state (memory) to process variable length sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition.

The term "recurrent neural network" is used indiscriminately to refer to two broad classes of networks with a similar general structure, where one is finite impulse and the other is infinite impulse. Both classes of networks exhibit temporal dynamic behavior. A finite impulse recurrent network is a directed acyclic graph that can be unrolled and replaced with a strictly feedforward neural network, while an infinite impulse recurrent network is a directed cyclic graph that cannot be unrolled.

Both finite impulse and infinite impulse recurrent networks can have additional stored states, and the storage can be under direct control by the neural network. The storage can also be replaced by another network or graph, if that incorporates time delays or has feedback loops. Such controlled states are referred to as gated state or gated memory, and are part of long short-term memory networks (LSTMs) and gated recurrent units. This is also called Feedback Neural Network.

Recurrent Neural Network remembers the past and it's decisions are influenced by what it has learnt from the past. Note: Basic feed forward networks "remember" things too, but they remember things they learnt during training. For example, an image classifier learns what a "1" looks like during training and then uses that knowledge to classify things in production.

While RNNs learn similarly while training, in addition, they remember things learnt from prior input(s) while generating output(s). It's part of the network.

RNNs can take one or more input vectors and produce one or more output vectors and the output(s) are influenced not just by weights applied on inputs like a regular NN, but also by a "hidden" state

vector representing the context based on prior input(s)/output(s). So, the same input could produce a different output depending on previous inputs in the series.

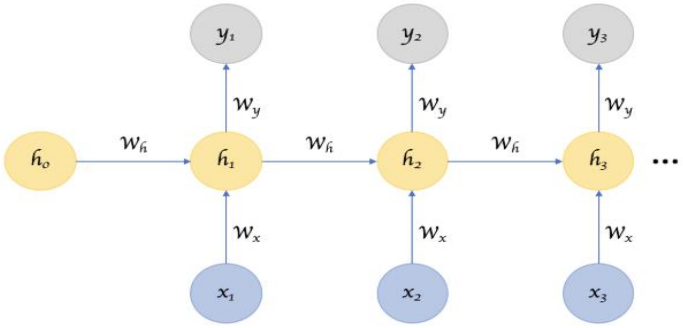


Figure 1: A Recurrent Neural Network, with a hidden state that is meant to carry pertinent information from one input item in the series to others.

There are following **Type of RNN:**

Deep RNNs: While it's good that the introduction of hidden state enabled us to effectively identify the relationship between the inputs, is there a way we can make a RNN "deep" and gain the multi-level abstractions and representations we gain through "depth" in a typical neural network?

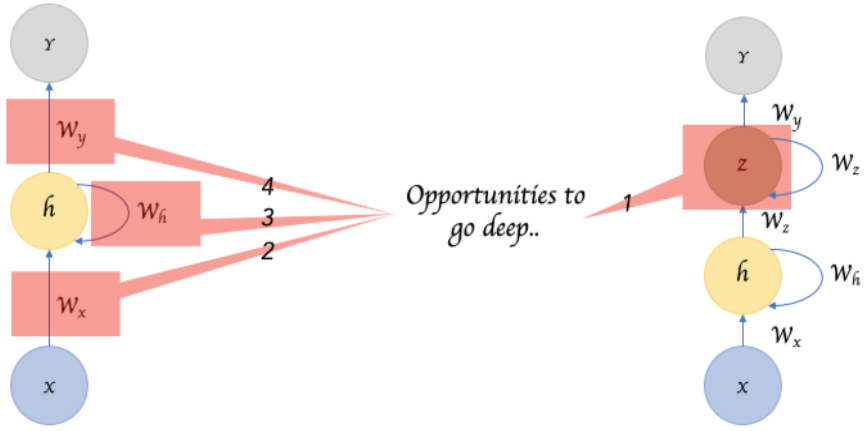


Figure 2: We can increase depth in three possible places in a typical RNN

Here are four possible ways to add depth. (1) Perhaps the most obvious of all, is to add hidden states, one on top of another, feeding the output of one to the next. (2) We can also add additional nonlinear hidden layers between input to hidden state (3) We can increase depth in the hidden to hidden transition (4) We can increase depth in the hidden to output transition.

Bidirectional RNNs:

Sometimes it's not just about learning from the past to predict the future, but we also need to look into the future to fix the past. In speech recognition and handwriting recognition tasks, where there could be considerable ambiguity given just one part of the input, we often need to know what's coming next to better understand the context and detect the present.

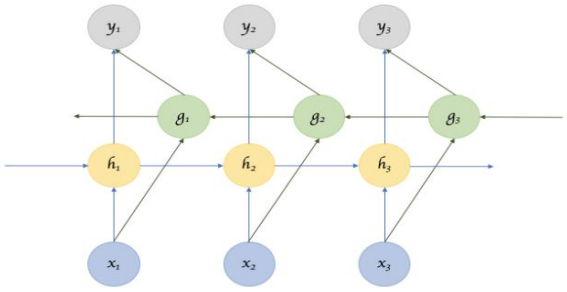


Figure 3: Bidirectional RNNs

This does introduce the obvious challenge of how much into the future we need to look into, because if we have to wait to see all inputs then the entire operation will become costly. And in cases like speech recognition, waiting till an entire sentence is spoken might make for a less compelling use case. Whereas for NLP tasks, where the inputs tend to be available, we can likely consider entire sentences all at once. Also, depending on the application, if the sensitivity to immediate and closer neighbors is higher than inputs that come further away, a variant that looks only into a limited future/past can be modeled.

Recursive Neural Networks:

A recurrent neural network parses the inputs in a sequential fashion. A recursive neural network is similar to the extent that the transitions are repeatedly applied to inputs, but not necessarily in a

sequential fashion. Recursive Neural Networks are a more general form of Recurrent Neural Networks. It can operate on any hierarchical tree structure. Parsing through input nodes, combining child nodes into parent nodes and combining them with other child/parent nodes to create a tree like structure. Recurrent Neural Networks do the same, but the structure there is strictly linear. i.e. weights are applied on the first input node, then the second, third and so on.

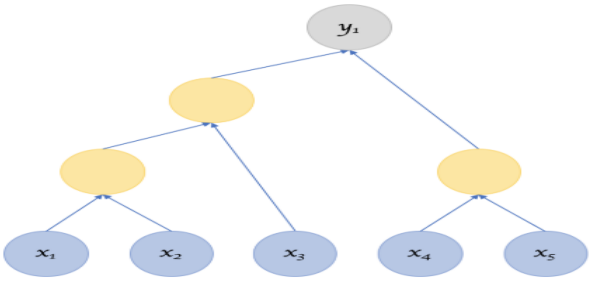


Figure 4: Recursive Neural Net

Long short-term memory (LSTM):

Long Short-Term Memory (LSTM) networks are a modified version of recurrent neural networks, which makes it easier to remember past data in memory. The vanishing gradient problem of RNN is resolved here. LSTM is well-suited to classify, process and predict time series given time lags of unknown duration. It trains the model by using back-propagation. In an LSTM network, three gates are present:

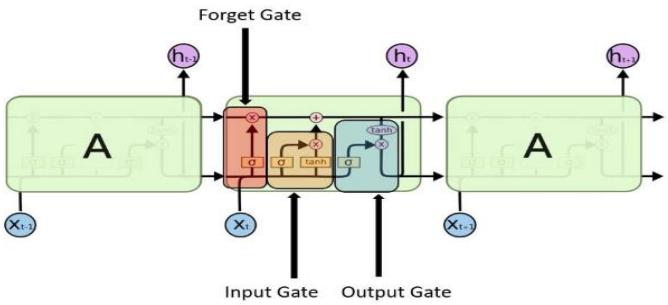


Figure: 5 LSTM Gates

Input gate — discover which value from input should be used to modify the memory. **Sigmoid** function decides which values to let through **0,1.** and **tanh** function gives weightage to the values which are passed deciding their level of importance ranging from-**1** to **1.**

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

Input gates

2. Forget gate — discover what details to be discarded from the block. It is decided by the **sigmoid function.** it looks at the previous state(**ht-1**) and the content input(**Xt**) and outputs a number between **0(***omit this*)and **1(***keep this***)**for each number in the cell state **Ct−1.**

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

Forget gate

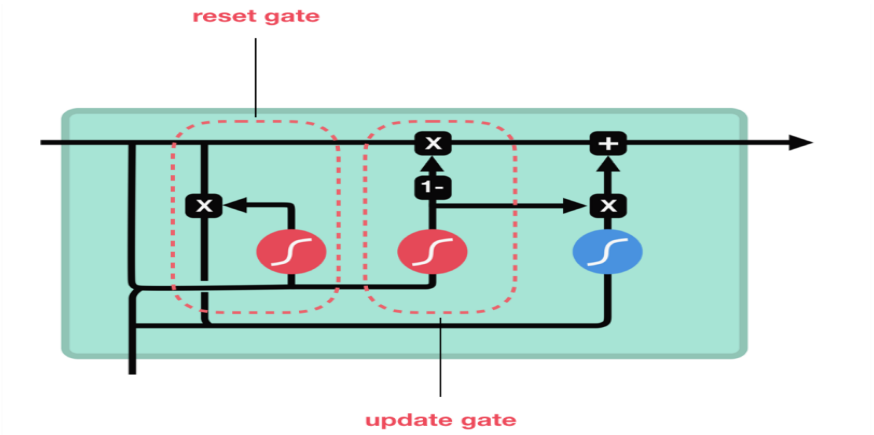
3. Output gate — the input and the memory of the block is used to decide the output. **Sigmoid** function decides which values to let through **0,1.** and **tanh** function gives weightage to the values which are passed deciding their level of importance ranging from-**1** to **1** and multiplied with output of **Sigmoid.**

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

Out put gate

Gated Recurrent Units (GRU):

The GRU is the newer generation of Recurrent Neural networks and is pretty similar to an LSTM. GRU's got rid of the cell state and used the hidden state to transfer information. It also only has two gates, a reset gate and update gate.



GRUcell and it's gates

Update Gate

The update gate acts similar to the forget and input gate of an LSTM. It decides what information to throw away and what new information to add.

Reset Gate

The reset gate is another gate is used to decide how much past information to forget.

And that's a GRU. GRU's has fewer tensor operations; therefore, they are a little speedier to train then LSTM's. There isn't a clear winner which one is better. Researchers and engineers usually try both to determine which one works better for their use case.

The GRU is like a long short-term memory (LSTM) with forget gate but has fewer parameters than LSTM, as it lacks an output gate. GRU's performance on certain tasks of polyphonic music modeling and speech signal modeling was found to be similar to that of LSTM. GRUs have been shown to exhibit even better performance on certain smaller datasets.

LSTM ’s and GRU’s were created as the solution to short-term memory

The Problem, Short-term Memory

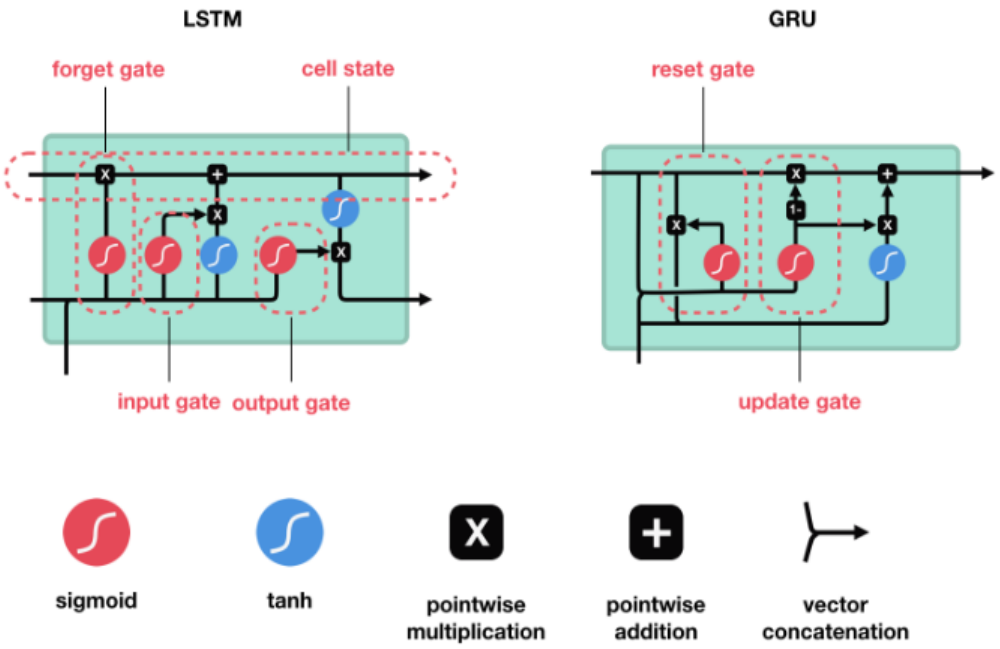
Recurrent Neural Networks suffer from short-term memory. If a sequence is long enough, they’ll have a hard time carrying information from earlier time steps to later ones. So if you are trying to process a paragraph of text to do predictions, RNN’s may leave out important information from the beginning.

During back propagation, recurrent neural networks suffer from the vanishing gradient problem. Gradients are values used to update a neural networks weight. The vanishing gradient problem is when the gradient shrinks as it back propagates through time. If a gradient value becomes extremely small, it doesn’t contribute too much learning.

So in recurrent neural networks, layers that get a small gradient update stops learning. Those are usually the earlier layers. So because these layers don’t learn, RNN’s can forget what it seen in longer sequences, thus having a short-term memory.

LSTM’s and GRU’s as a solution

LSTM ’s and GRU’s were created as the solution to short-term memory. They have internal mechanisms called gates that can regulate the flow of information.



These gates can learn which data in a sequence is important to keep or throw away. By doing that, it can pass relevant information down the long chain of sequences to make predictions. Almost all state-of-the-art results based on recurrent neural networks are achieved with these two networks. LSTM's and GRU's can be found in speech recognition, speech synthesis, and text generation.

What is Beam search?

beam search is a heuristic search algorithm that explores a graph by expanding the most promising node in a limited set. Beam search is an optimization of best-first search that reduces its memory requirements. Best-first search is a graph search which orders all partial solutions (states) according to some heuristic. But in beam search, only a predetermined number of best partial solutions are kept as candidates.

The beam search algorithm selects multiple alternatives for an input sequence at each timestep based on conditional probability. The number of multiple alternatives depends on a parameter called Beam Width B. At each time step, the beam search selects B number of best alternatives with the highest probability as the most likely possible choices.

Uses:

A beam search is most often used to maintain tractability in large systems with insufficient amount of memory to store the entire search tree. For example, it has been used in many machine translation systems. (the state of the art now primarily uses neural machine translation based methods). To select the best translation, each part is processed, and many different ways of translating the words appear. The top best translations according to their sentence structures are kept, and the rest are discarded. The translator then evaluates the translations according to a given criterion, choosing the translation which best keeps the goals.

Reinforcement learning and Frame work: is a decision-making system its ability to takes decision with the help of reward and penalty over environment. Agent is a program code and also called decision making. RL is based on self-learning phenomenon that have not prior knowledge about system.

After offset of trial and error runs, it should learn the best policy, which is the sequence of actions that maximize the total reward. This is achieved entire goals through optimal action.

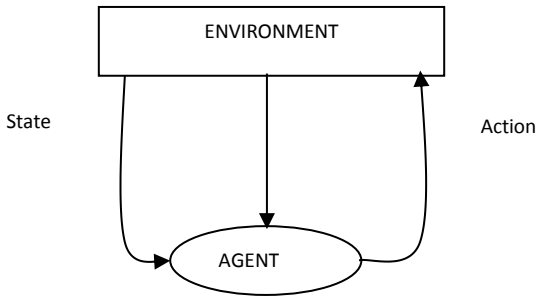


Fig.1 The agent interacts with environment at any state of the environment, the agent takes action that changes the state and returns a reward.

It offers a general structure and several methods to make it improve its behavior. RL is not That In case of large search spaces or large data sets our existing learning algorithm required large storage or look up table. Matter is that how to memorize or reduce or optimize look-up-table.

We are analyzing and study that how to optimize look up table. It can be possible with the help of DCT. It is also helpful for exponential data storage in matrix form.

In RL, the optimal action output is not provided to the learner, which is in contrast to supervised learning methods. Therefore, the learner is able to change current policy and commutative rewards during trial and error interaction. In recent years, among most reactive control methods,

reinforcement learning has been broadly applied into robot navigation field in unknown environment because of its self-learning and on line learning abilities. RL is unsupervised or on-line learning method, in which agent is given feedback via reward and penalty over environment. The basic idea is that action correlated with high reward and due to higher iterative probability required much storage space, it is required for mapping the situation, applied by hidden Marko model (HMM). While those correlated with low reward have lower iterative probability, Used method as a incremental and real time learning method.

Reinforcement learning focuses on, how to agent learn without prior knowledge of the system. We can say that it may applied over un-certain environment through commutative reward, it should be maximized. In consecutive learning agent should require customized expected reward which help for selection of the optimal policy and chooses optimal policy among them. After select policy apply action over environment.

There are many unsolved problems that computers could solve if the appropriate software existed.

Reinforcement learning (**RL**) is an area of machine learning concerned with how software agents ought to take actions in an environment in order to maximize the notion of cumulative reward. Reinforcement learning is one of three basic machine learning paradigms, alongside supervised learning and unsupervised learning.

Reinforcement learning differs from supervised learning in not needing labelled input/output pairs be presented, and in not needing sub-optimal actions to be explicitly corrected. Instead the focus is on finding a balance between exploration (of uncharted territory) and exploitation (of current knowledge).

Reinforcement learning, due to its generality, is studied in many other disciplines, such as game theory, control theory, operations research, information theory, simulation-based optimization, multi-agent systems, swarm intelligence, statistics and genetic algorithms. reinforcement learning is called approximate dynamic programming, or neuro-dynamic programming. The problems of interest in reinforcement learning have also been studied in the theory of optimal control, which is concerned mostly with the existence and characterization of optimal solutions, and algorithms for their exact computation, and less with learning or approximation, particularly in the absence of a mathematical model of the environment.

In economics and game theory, reinforcement learning may be used to explain how equilibrium may arise under bounded rationality.

Markov decision process(MDP):

A **Markov decision process** (**MDP**) is a discrete time stochastic control process. It provides a mathematical framework for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker. MDPs are useful for studying optimization problems solved via dynamic programming and reinforcement learning. MDPs were known also Dynamic Programming and Markov Processes. They are used in many disciplines, including robotics, automatic control, economics and manufacturing. The name of MDPs comes from the Russian mathematician Andrey Markov as they are an extension of the Markov chains.

A stochastic process has the Markov property if the conditional probability distribution of future states of the process (conditional on both past and present values) depends only upon the present state; that is, given the present, the future does not depend on the past. A process with this property is said to be **Markovian** or a **Markov process**. The most famous Markov process is a Markov chain. Brownian motion is another well-known Markov process.

A **Bellman equation**, named after Richard E. **Bellman**, is a necessary condition for optimality associated with the mathematical optimization method known as dynamic programming. However, the term '**Bellman equation**' usually refers to the dynamic programming **equation** associated with discrete-time optimization problems.

To understand the Bellman equation, several underlying concepts must be understood. First, any optimization problem has some objective: minimizing travel time, minimizing cost, maximizing profits, maximizing utility, etc. The mathematical function that describes this objective is called the objective function.

Dynamic programming breaks a multi-period planning problem into simpler steps at different points in time. Therefore, it requires keeping track of how the decision situation is evolving over time. The information about the current situation that is needed to make a correct decision is called the state. For example, to decide how much to consume and spend at each point in time, people would need to The variables chosen at any given point in time are often called the control variables. For example, given their current wealth, people might decide how much to consume now. Choosing the control variables now may be equivalent to choosing the next state; more generally, the next state is affected by other factors in addition to the current control. For example, in the simplest case, today's wealth

(the state) and consumption (the control) might exactly determine tomorrow's wealth (the new state), though typically other factors will affect tomorrow's wealth too.

The dynamic programming approach describes the optimal plan by finding a rule that tells what the controls should be, given any possible value of the state.

A **Bellman equation** is represented as :

$$\max_{a_0}\left\{F(x_0,a_0)+\beta\left[\max_{\{a_t\}_{t=1}^{\infty}}\sum_{t=1}^{\infty}\beta^{t-1}F(x_t,a_t):a_t\in\Gamma(x_t),\ x_{t+1}=T(x_t,a_t),\ \forall t\geq 1\right]\right\}$$

subject to the constraints

$$a_0\in\Gamma(x_0),\ x_1=T(x_0,a_0).$$

Value Iteration and Policy Iteration:

In previous sections the environment and the reinforcement function are discussed. However, the issue of how the agent learns to choose "good" actions or even how we might measure the utility of an action is not explained. First, two terms are defined. Policy determines which action should be performed in each state; a policy is a mapping from states to actions. The value of a state is defined as the sum of the reinforcements received when starting in that state and following some fixed policy to a terminal state. The optimal policy would therefore be the mapping from states to actions that maximizes the sum of the reinforcements.

The value function is a mapping from states to state values and can be approximated using any type of function approximator (e.g., multilayered perceptron, memory-based system, radial basis functions, look-up table, etc.).

An example of a value function can be seen using a simple Markov decision process with 16 states. The state space can be visualized using a 4x4 grid in which each square represents a state.

The reinforcement function is (-1) everywhere (i.e., the agent receives a reinforcement of (-1) on each transition). There are 4 actions possible in each state: north, south, east, and west. The goal states are the upper left corner and the lower right corner. The value function for the random policy is shown in Figure 1. For each state the random policy randomly chooses one of the four possible actions. The numbers in the states represent the expected values of the states. For example, when starting in the lower left corner and following a random policy, on average there will be 22 transitions to other states before the terminal state is reached.

0	**-14**	**-20**	**-22**
-14	**-18**	**-22**	**-20**
-20	**-22**	**-18**	**-14**
-22	**-20**	**-14**	**0**

Figure 1

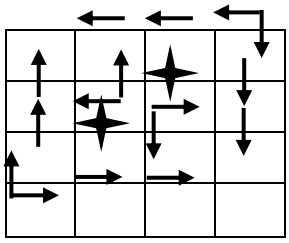


Figure 3

0	**-1**	**-2**	**-3**
-1	**-2**	**-3**	**-2**
-2	**-3**	**-2**	**-1**
-3	**-2**	**-1**	**0**

Figure 2

The optimal value function is shown in Figure 2. Again, starting in the lower left corner, calculating the sum of the reinforcements when performing the optimal policy (the policy that will maximize the sum of the reinforcements), the value of that state is -3 because it takes only 3 transitions to reach a terminal state. If we are given the optimal value function, then it becomes a trivial task to extract the optimal policy. For example, one can start in any state in Figure 2 and simply choose the action that maximizes the immediate reinforcement received. In other words, one can perform a one level deep breadth-first search over actions to find the action that will maximize the immediate reward. The optimal policy for the value function shown in Figure 2 is given in Figure 3.

Approximating the Value Function:

Reinforcement learning is a difficult problem because the learning system may perform an action and not be told whether that action was good or bad. For example, a learning auto-pilot program might be given control of a simulator and told not to crash. It will have to make many decisions each second and then, after acting on thousands of decisions, the aircraft may crash due to …….. What should the system learn from this experience? Which of its many actions was responsible for the crash? Assigning blame to individual

actions is the problem that makes reinforcement learning difficult. Surprisingly, there is a solution to this problem. It is based on a field of mathematics called dynamic programming, and it involves just two basic principles. First, if an action causes something bad to happen immediately, such as crashing the plane, then the system learns not to do that action in that situation again. So whatever action the system performed one millisecond before the crash, it will avoid doing in the future. But that principle doesn't help for all the earlier actions which didn't lead to immediate disaster.

The second principle is that if all the actions in a certain situation lead to bad results, then that situation should be avoided. So if the system has experienced a certain combination of altitude and airspeed at many different times, whereby trying a different action each time, and all actions led to something bad, then it will learn that the situation itself is bad. This is a powerful principle, because the learning system can now learn without crashing. In the future, any time it chooses an action that leads to this particular situation, it will immediately learn that particular action is bad, without having to wait for the crash. By using these two principles, a learning system can learn to fly a plane, control a robot, or do any number of tasks. It can first learn on a simulator, then fine tune on the actual system. This technique is generally referred to as dynamic programming, and a slightly closer analysis will reveal how dynamic programming can generate the optimal value function.

Difference between policy iteration and Value iteration:

In policy iteration algorithms, you start with a random policy, then find the value function of that policy (policy evaluation step), then find a new (improved) policy based on the previous value function, and so on. In this process, each policy is guaranteed to be a strict improvement over the previous one (unless it is already optimal). Given a policy, its value function can be obtained using the Bellman operator.

In value iteration, you start with a random value function and then find a new (improved) value function in an iterative process, until reaching the optimal value function. Notice that you can derive easily the optimal policy from the optimal value function. This process is based on the optimality Bellman operator.

In some sense, both algorithms share the same working principle, and they can be seen as two cases of the generalized policy iteration. However, the optimality Bellman operator contains a max operator, which is nonlinear and, therefore, it has different features. In addition, it's possible to use hybrid methods between pure value iteration and pure policy iteration.

Actor-Critics aim to take advantage of all the good stuff from both value-based and policy-based while eliminating all their drawbacks. And how do they do this? The principal

idea is to split the **model** in two: one for computing an action based on a state and another one to produce the Q values of the action.

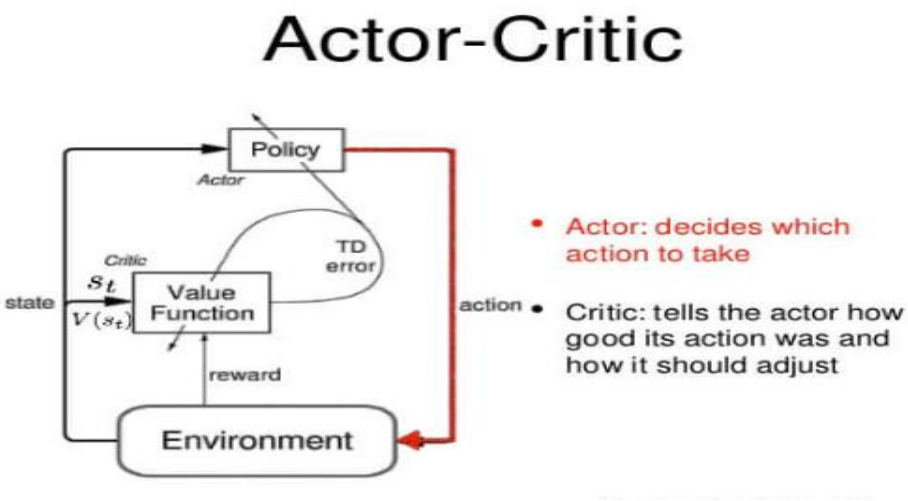


Figure: Actor-critic model

Q-learning, SARSA

Q-Learning: Q-learning is a form of model free reinforcement learning. It provides agent with the capability of learning to act optimally in Markov domains by experiencing action consequences. Agent selects optimal policy through iteration (Q function), Q learning usually stores Q value (S, A) relative with every state action in a lookup table . In Q learning there are learning steps repeat in every episode till n^{th} time:

Evaluation of Q value:

Q values of each agent are evaluated by the sum of the rewards which the agent obtains during the previous one episode using this equation.

$Q(s_n, a_n)=r_{n+1}+\gamma \max_{an+1} Q(s_{n+1},a_{n+1})$

And simply use this as an assignment to update $Q(s_n, a_n)$ When in state s_n, we choose action a_n by one of the stochastic strategies we saw earlier , which returns a reward r_{n+1} and takes us to state s_{n+1}. We then update the value of previous action as.

$Q`(s_n, a_n)=r_{n+1}+\gamma \max_{an+1} Q`(s_{n+1},a_{n+1})$

Where the des denote the value is an estimate $Q`(s_{n+1},a_{n+1})$ is a later value and has higher chance of being correct

γ : discounted rate parameter that reduces the influence of future expected rewards($0<\gamma<1$)

r : reward

s_n: next step

a_n: action taken in

s_0: initial step

α= learning rate

a_0=initial action

Q value is updated whenever an agent takes an action in the individual learning, Q values at and shortly after beginning rather of the episode is rather different from those at the end of the episode. Even if a new episode begins with the Q values at the end of the previous episode are not necessarily taken and the same rewards are not necessarily obtained. The rewards obtained by using the Q values at and shortly after the beginning are considered to have only a little relation with the Q values at the end. Thus such rewards are

discounted and the discounted and the discounted results are summed up. Therefore, we define the evaluated value E for the Q value at the end each episode .
Where N is the number of actions in the episode, r_k is the reward for the k^{th} action, and d(<1) is the discount parameter , The definition (1) could bring that the larger the evaluated value E for Q-values is the superior the Q-value are:
Update Q value In order to update Q-values through exchanging the information among the agents, Best state action value method, If the superior Q values are copied to those for each of the other agents, each agent can improve its Q-values in future episodes, according to this idea the Q value $Q_i(s, a)$ of agent I for action a in state s is updated by

$$Q_i(s, a) \leftarrow Q^{best}(s, a)$$

Where $Q^{best}(s, a)$ is the Q-value which is evaluated superior to that of any other agent , and is called the best Q-value.

Average state action value method in best, the Q-values which are not best are discarded, and replaced with the best Q-values. However, they are not necessarily worthless, because they are updated by agents through the individual learning. Optimal Q-values may not be found [5]. These drawbacks of BEST, in AVE each agent updates its Q-values by averaging its current Q-value and the best Q-value for each state action.

SARSA (State Action Reward State Action):

SARSA algorithm is a slight variation of the popular Q-Learning algorithm. For a learning agent in any Reinforcement Learning algorithm it's policy can be of two types:-

1. **On Policy:** In this, the learning agent learns the value function according to the current action derived from the policy currently being used.
2. **Off Policy:** In this, the learning agent learns the value function according to the action derived from another policy.

Q-Learning technique is an **Off-Policy** technique and uses the greedy approach to learn the Q-value. SARSA technique, on the other hand, is an **On Policy** and uses the action performed by the current policy to learn the Q-value. the difference of the update statements for each technique: -

1. **Q-Learning:**
$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t))$$
2. **SARSA:**
$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha(r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t))$$

Here, the update equation for SARSA depends on the current state, current action, reward obtained, next state and next action. This observation lead to the naming of the learning technique as SARSA stands for **State Action Reward State Action** which symbolizes the tuple (s, a, r, s', a').

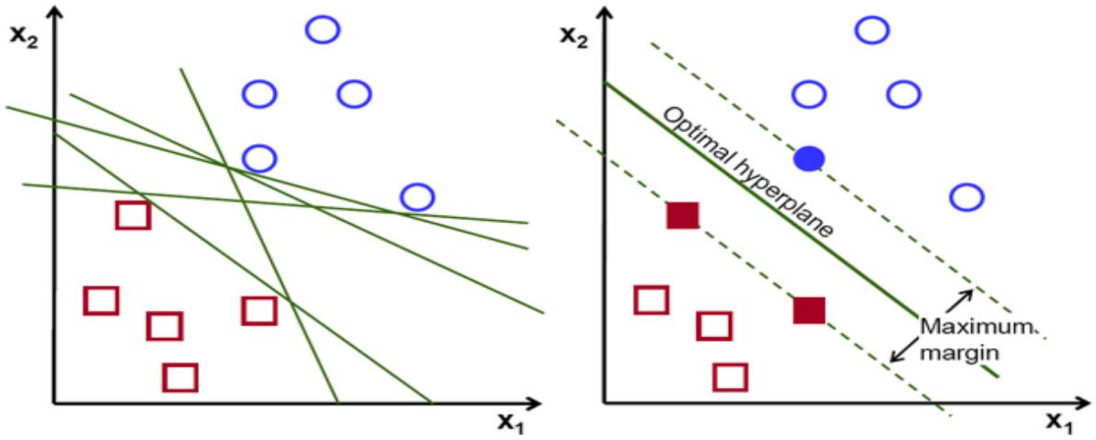
Support Vector Machine (SVM):

The objective of the support vector machine algorithm is to fined a hyperplane in an N-dimensional space that classify the data point.

support-vector machines (SVMs) are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on the side of the gap on which they fall.

In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces.

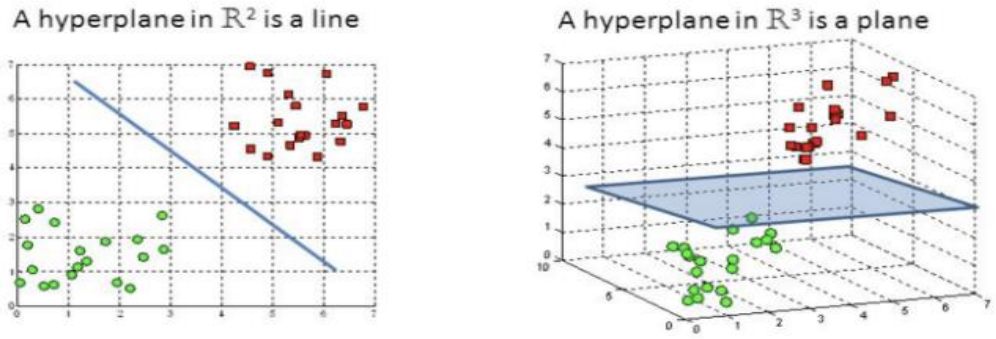
When data are unlabeled, supervised learning is not possible, and an unsupervised learning approach is required, which attempts to find natural clustering of the data to groups, and then map new data to these formed groups.



Possible hyperplanes

To separate the two classes of data points, there are many possible hyperplanes that could be chosen. Our objective is to find a plane that has the maximum margin, i.e the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence.

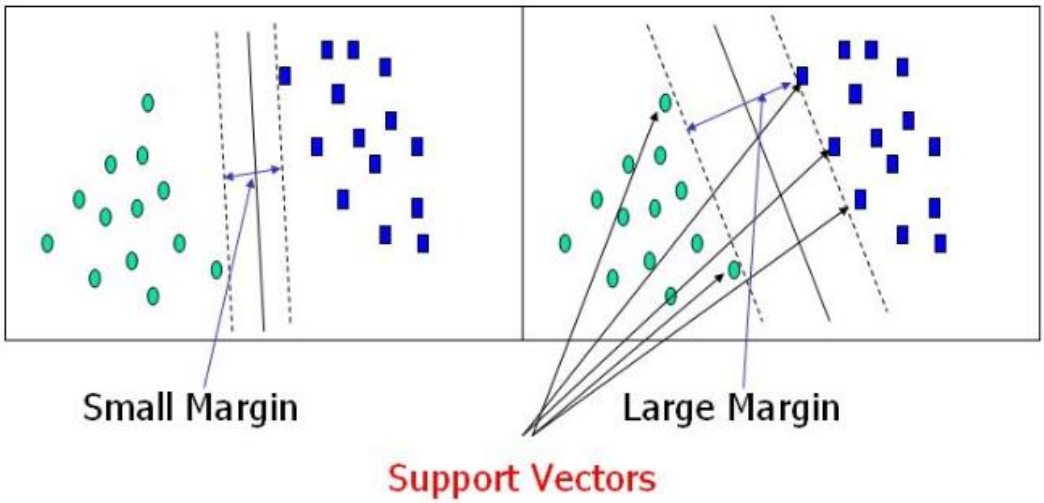
Hyperplanes and Support Vectors:



Hyperplanes in 2D and 3D feature space

Hyperplanes are decision boundaries that help classify the data points. Data points falling on either side of the hyperplane can be attributed to different classes. Also, the dimension of the hyperplane depends upon the number of features. If the number of input features is 2, then the hyperplane is just a line. If the number of

input features is 3, then the hyperplane becomes a two-dimensional plane. It becomes difficult to imagine when the number of features exceeds 3.



Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane. Using these support vectors, we maximize the margin of the classifier. Deleting the support vectors will change the position of the hyperplane. These are the points that help us build our SVM.

Large Margin Intuition

In logistic regression, we take the output of the linear function and squash the value within the range of [0,1] using the sigmoid function. If the squashed value is greater than a threshold value (0.5) we assign it a label 1, else we assign it a label 0. In SVM, we take the output of the linear function and if that output is greater than 1, we identify it with one class and if the output is -1, we identify is with another class. Since the threshold values are changed to 1 and -1 in SVM, we obtain this reinforcement range of values ([-1,1]) which acts as margin.

Cost Function and Gradient Updates

In the SVM algorithm, we are looking to maximize the margin between the data points and the hyperplane. The loss function that helps maximize the margin is hinge loss.

$$c(x, y, f(x)) = \begin{cases} 0, & \text{if } y * f(x) \geq 1 \\ 1 - y * f(x), & \text{else} \end{cases} \quad c(x, y, f(x)) = (1 - y * f(x))$$

Hinge loss function n (function on left can be represented as a function on the right)

The cost is 0 if the predicted value and the actual value are of the same sign. If they are not, we then calculate the loss value. We also add a regularization parameter the cost function. The objective of the regularization parameter is to balance the margin maximization and loss. After adding the regularization parameter, the cost functions look as below.

$$min_w \lambda \parallel w \parallel^2 + \sum_{i=1}^{n} (1 - y_i \langle x_i, w \rangle)_+$$

Loss function for SVM

Now that we have the loss function, we take partial derivatives with respect to the weights to find the gradients. Using the gradients, we can update our weights.

$$\frac{\delta}{\delta w_k} \lambda \parallel w \parallel^2 = 2\lambda w_k$$

$$\frac{\delta}{\delta w_k} (1 - y_i \langle x_i, w \rangle)_+ = \begin{cases} 0, & \text{if } y_i \langle x_i, w \rangle \geq 1 \\ -y_i x_{ik}, & \text{else} \end{cases}$$

Gradients

When there is no misclassification, i.e our model correctly predicts the class of our data point, we only have to update the gradient from the regularization parameter.

$$w = w - \alpha \cdot (2\lambda w)$$

Gradient Update — No misclassification

When there is a misclassification, i.e our model make a mistake on the prediction of the class of our data point, we include the loss along with the regularization parameter to perform gradient update.

$$w = w + \alpha \cdot (y_i \cdot x_i - 2\lambda w)$$

Gradient Update Misclassification

Bayesian learning:

So far, we have discussed Bayes' theorem and gained an understanding of how we can apply Bayes' theorem to test our hypotheses. However, when using single point estimation techniques such as MAP, we will not be able to exploit the full potential of Bayes' theorem. We used single values to explain each term in Bayes' theorem to simplify my explanation of Bayes' theorem.

Bayesian learning, we are dealing with random variables that have probability distributions. Let us try to understand why using exact point estimations can be misleading in probabilistic concepts.

Bayes' Theorem:

Bayes' theorem describes how the conditional probability of an event or a hypothesis can be computed using evidence and prior knowledge. It is similar to concluding that our code has no bugs given the evidence that it has passed all the test cases, including our prior belief that we have rarely observed any bugs in our code. However, this intuition goes beyond that simple hypothesis test where there are multiple events or hypotheses involved (let us not worry about this for the moment).

$$P(\theta|X) = \frac{P(X|\theta)P(\theta)}{P(X)}$$

Maximum a Posteriori (MAP):

We can use MAP to determine the valid hypothesis from a set of hypotheses. According to MAP, the hypothesis that has the maximum posterior probability is considered as the valid hypothesis.

$$\begin{aligned}\theta_{MAP} &= argmax_{\theta} P(\theta_i|X) \\ &= argmax_{\theta} \left(\frac{P(X|\theta_i)P(\theta_i)}{P(X)} \right)\end{aligned}$$

Binomial Likelihood:

The likelihood for the coin flip experiment is given by the probability of observing heads out of all the coin flips given the fairness of the coin.

As we have defined the fairness of the coins using the probability of observing heads for each coin flip, we can define the probability of observing heads or tails given the fairness of the coin P(y|theta) where y = 1 for observing heads and y = 0, observing tails. Accordingly:

$$P(y = 1|\theta) = \theta$$
$$P(y = 0|\theta) = (1 - \theta)$$

Now that we have defined two conditional probabilities for each outcome above, let us now try to find the P(Y=y| theta) joint probability of observing heads or tails:

$$P(Y = y|\theta) = \begin{cases} \theta, \text{ if } y = 1 \\ 1 - \theta, \text{ otherwise} \end{cases}$$

We can rewrite the above expression in a single expression as follows:

$$P(Y = y|\theta) = \theta^y \times (1 - \theta)^{1-y}$$

The above equation represents the likelihood of a single test coin flip experiment. Interestingly, the likelihood function of the single coin flip experiment is similar to the Bernoulli probability distribution. The Bernoulli distribution is the probability distribution of a single trial experiment with only two opposite outcomes.

Binomial probability distribution as shown below:

$$P(k, N|\theta) = \binom{N}{k} \theta^k (1 - \theta)^{N-k}$$

Beta Prior Distribution

The prior distribution is used to represent our belief about the hypothesis based on our past experiences. We can choose any distribution for the prior, if it represents our belief regarding the fairness of the coin. For this example, we use Beta distribution to represent the prior probability distribution as follows:

$$P(\theta) = \frac{\theta^{\alpha-1}(1 - \theta)^{\beta-1}}{B(\alpha, \beta)}$$

Application of machine learning in computer vision:

Machine learning has improved computer vision about recognition and tracking. It offers effective methods for acquisition, image processing, and object focus which are used in computer vision. There are following field where computer Vision applied:

1. Image Classification
2. Image Classification with Localization
3. Object Detection
4. Object Segmentation
5. Image Style Transfer
6. Image Colorization
7. Image Reconstruction
8. Image Super-Resolution
9. Image Synthesis
10. Other Problems

Image Classification

Image classification involves assigning a label to an entire image or photograph.

This problem is also referred to as object classification and perhaps more generally as image recognition, although this latter task may apply to a much broader set of tasks related to classifying the content of images.
Some examples of image classification include:

- Labeling an x-ray as cancer or not (binary classification).
- Classifying a handwritten digit (multiclass classification).
- Assigning a name to a photograph of a face (multiclass classification).

Image Classification with Localization

Image classification with localization involves assigning a class label to an image and showing the location of the object in the image by a bounding box (drawing a box around the object).

This is a more challenging version of image classification.

Some examples of image classification with localization include:

- Labeling an x-ray as cancer or not and drawing a box around the cancerous region.
- Classifying photographs of animals and drawing a box around the animal in each scene.

Object Detection

Object detection is the task of image classification with localization, although an image may contain multiple objects that require localization and classification.

This is a more challenging task than simple image classification or image classification with localization, as often there are multiple objects in the image of different types.

Often, techniques developed for image classification with localization are used and demonstrated for object detection.

Some examples of object detection include:

- Drawing a bounding box and labeling each object in a street scene.
- Drawing a bounding box and labeling each object in an indoor photograph.
- Drawing a bounding box and labeling each object in a landscape.

Object Segmentation

Object segmentation, or semantic segmentation, is the task of object detection where a line is drawn around each object detected in the image. Image segmentation is a more general problem of spitting an image into segments. Object detection is also sometimes referred to as object segmentation. Unlike object detection that involves using a bounding box to identify objects, object segmentation identifies the specific pixels in the image that belong to the object. It is like a fine-grained localization.

Application of machine learning in computer speech processing:

Machine learning uses iterative algorithms to learn from data and allows the computer to find information, hidden values that are not explicitly

programmed. Machine Learning systems can quickly apply knowledge and training from large datasets to perform face recognition, speech recognition.

Introducing speech recognition digital assistants

Digital assistants are designed to help people perform or complete basic tasks and respond to queries. With the ability to access information from vast databases and various digital sources, these robots help to solve problems in real time, enhancing the user experience and human productivity.

Popular digital assistants, include:

- Amazon's Alexa
- Apple's Siri
- Google's Google Assistant
- Microsoft's Cortana

Applications of speech recognition technology:

1. In the workplace

Speech recognition technology in the workplace has evolved into incorporating simple tasks to increase efficiency, as well as beyond tasks that have traditionally needed humans, to be performed.

Examples of office tasks digital assistants are, or will be, able to perform:

- Search for reports or documents on your computer
- Create a graph or tables using data
- Dictate the information you want to be incorporated into a document
- Print documents on request
- Start video conferences
- Schedule meetings
- Record minutes
- Make travel arrangements

***2*. In banking**

The aim of the banking and financial industry is for speech recognition to reduce friction for the customer.[8] Voice-activated banking could largely reduce the need for human customer service, and lower employee costs. A personalized banking assistant could in return boost customer satisfaction and loyalty. How speech recognition could improve banking:

- Request information regarding your balance, transactions, and spending habits without having to open your cell phone
- Make payments
- Receive information about your transaction history

4. In Healthcare

In an environment where seconds are crucial and sterile operating conditions are a priority, hands-free, immediate access to information can have a significantly positive impact on patient safety and medical efficiency.

Benefits include:

- Quickly finding information from medical records
- Nurses can be reminded of processes or given specific instructions
- Nurses can ask for administrative information, such as the number of patients on a floor and the number of available units
- At home, parents can ask for common symptoms of diseases, when they should go to the doctor, and how to look after a sick child
- Less paperwork
- Less time inputting data
- Improved workflows

The most significant concern using speech recognition in healthcare is the content the digital assistant has access to. It has been recognized that the content will need to be supplied and validated by recognized medical institutions, in order for it to be a viable option in this field.

Application of machine learning in computer natural language processing:

1. **Text Classification and Categorization**

Text classification is an essential part in many applications, such as web searching, information filtering, language identification, readability assessment, and sentiment analysis. Neural networks are actively used for these tasks.

2. Semantic Parsing and Question Answering

Question Answering systems automatically answer different types of questions asked in natural languages including definition questions, biographical questions, multilingual questions, and so on. Neural networks usage makes it possible to develop high performing question answering systems.

3. Language Generation and Multi-document Summarization

Natural language generation has many applications such as automated writing of reports, generating texts based on analysis of retail sales data, summarizing electronic medical records, producing textual weather forecasts from weather data, and even producing jokes.

4. Machine Translation

Machine translation software is used around the world despite its limitations. In some domains, the quality of translation is not good. To improve the results researchers, try different techniques and models, including the neural network approach. The purpose of Neural-based Machine Translation for Medical Text Domain study is to inspect the effects of different training methods on a Polish-English machine translation system used for medical data. To train neural and statistical network-based translation systems.

5. Spell Checking

Most text editors let users check if their text contains spelling mistakes. Neural networks are now incorporated into many spell-checking tools.

6. Character Recognition

Character Recognition systems also have numerous applications like receipt character recognition, invoice character recognition, check character recognition, legal billing document character recognition, and so on.