RAJIV GANDHI PROUDYOGIKI VISHWAVIDYALAYA, BHOPAL

New Scheme Based On AICTE Flexible Curricula

Computer Science and Engineering, VI-Semester

CS601 Machine Learning

COURSE OUTCOMES:

After Completing the course student should be able to:

1. Apply knowledge of computing and mathematics to machine learning problems, models and algorithms;
2. Analyze a problem and identify the computing requirements appropriate for its solution;
3. Design, implement, and evaluate an algorithm to meet desired needs; and
4. Apply mathematical foundations, algorithmic principles, and computer science theory to the modeling and design of computer-based systems in a way that demonstrates comprehension of the trade-offs involved in design choices.

COURSE CONTENTS:

THEOTY:

Unit –I

Introduction to machine learning, scope and limitations, regression, probability, statistics and linear algebra for machine learning, convex optimization, data visualization, hypothesis function and testing, data distributions, data preprocessing, data augmentation, normalizing data sets, machine learning models, supervised and unsupervised learning.

Unit –II

Linearity vs non linearity, activation functions like sigmoid, ReLU, etc., weights and bias, loss function, gradient descent, multilayer network, backpropagation, weight initialization, training, testing, unstable gradient problem, auto encoders, batch normalization, dropout, L1 and L2 regularization, momentum, tuning hyper parameters,

Unit –III

Convolutional neural network, flattening, subsampling, padding, stride, convolution layer, pooling layer, loss layer, dance layer 1x1 convolution, inception network, input channels, transfer learning, one shot learning, dimension reductions, implementation of CNN like tensor flow, keras etc.

Unit –IV

Recurrent neural network, Long short-term memory, gated recurrent unit, translation, beam search and width, Bleu score, attention model, Reinforcement Learning, RL-framework, MDP, Bellman equations, Value Iteration and Policy Iteration, , Actor-critic model, Q-learning, SARSA

Unit –V

Support Vector Machines, Bayesian learning, application of machine learning in computer vision, speech processing, natural language processing etc, Case Study: ImageNet Competition

TEXT BOOKS RECOMMENDED:

1. Christopher M. Bishop, "Pattern Recognition and Machine Learning", Springer-Verlag New York Inc., 2nd Edition, 2011.
2. Tom M. Mitchell, "Machine Learning", McGraw Hill Education, First edition, 2017.
3. Ian Goodfellow and Yoshua Bengio and Aaron Courville, "Deep Learning", MIT Press, 2016

REFERENCE BOOKS:

1. Aurelien Geon, "Hands-On Machine Learning with Scikit-Learn and Tensorflow: Concepts, Tools, and Techniques to Build Intelligent Systems", Shroff/O'Reilly; First edition (2017).
2. Francois Chollet, "Deep Learning with Python", Manning Publications, 1 edition (10 January 2018).
3. Andreas Muller, "Introduction to Machine Learning with Python: A Guide for Data Scientists", Shroff/O'Reilly; First edition (2016).
4. Russell, S. and Norvig, N. "Artificial Intelligence: A Modern Approach", Prentice Hall Series in Artificial Intelligence. 2003.

PRACTICAL:

Different problems to be framed to enable students to understand the concept learnt and get hands-on on various tools and software related to the subject. Such assignments are to be framed for ten to twelve lab sessions.

Downloaded from www.rgpvnotes.in

Department of Computer Science and Engineering
Subject Notes
CS 601- Machine Learning
UNIT-I

Introduction to machine learning:

Machine learning is a tool for turning information into knowledge. Machine learning techniques are used to automatically find the valuable underlying patterns within complex data that we would otherwise struggle to discover. The hidden patterns and knowledge about a problem can be used to predict future events and perform all kinds of complex decision making.

Tom Mitchell gave a "well-posed" mathematical and relational definition that "A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

For Example:

A checkers learning problem:

Task(T): Playing checkers.

Performance measures (P): Performance of games won.

Training Experience (E): Playing practice games against itself.

Need For Machine Learning

- Ever since the technical revolution, we've been generating an immeasurable amount of data.
- With the availability of so much data, it is finally possible to build predictive models that can study and analyse complex data to find useful insights and deliver more accurate results.
- Top Tier companies such as Netflix and Amazon build such Machine Learning models by using tons of data in order to identify profitable opportunities and avoid unwanted risks.

Stream Tech Notes

ML Vs AI Vs DL

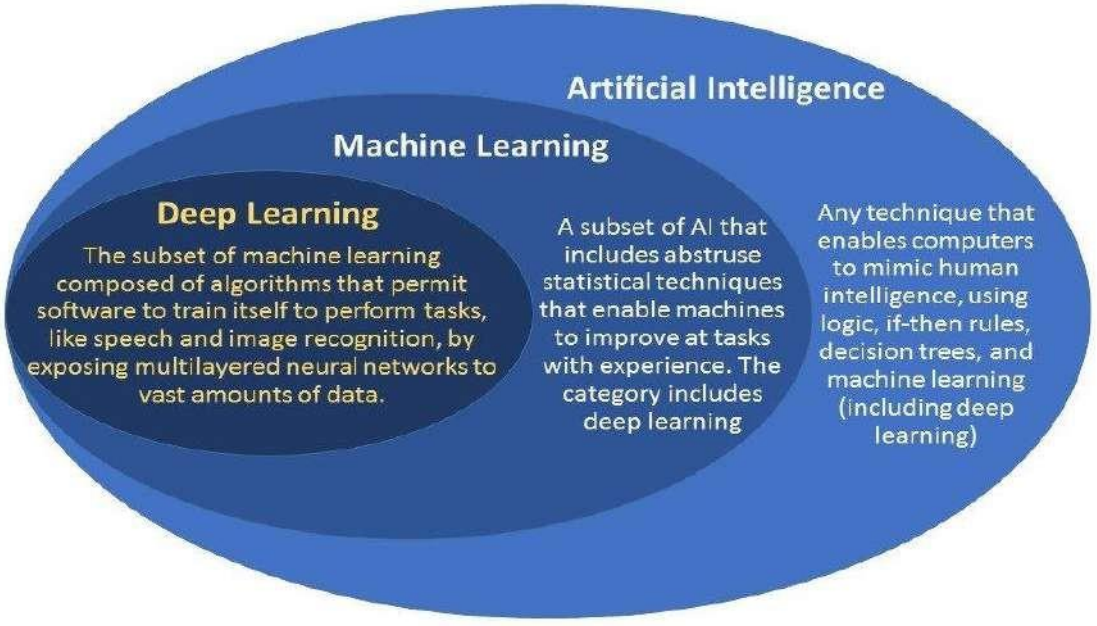


Figure: 1.1

Important Terms of Machine Learning

- **Algorithm:** A Machine Learning algorithm is a set of rules and statistical techniques used to learn patterns from data and draw significant information from it. It is the logic behind a Machine Learning model. An example of a Machine Learning algorithm is the Linear Regression algorithm.

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

- **Model:** A model is the main component of Machine Learning. A model is trained by using a Machine Learning Algorithm. An algorithm maps all the decisions that a model is supposed to take based on the given input, in order to get the correct output.
- **Predictor Variable:** It is a feature(s) of the data that can be used to predict the output.
- **Response Variable:** It is the feature or the output variable that needs to be predicted by using the predictor variable(s).
- **Training Data:** The Machine Learning model is built using the training data. The training data helps the model to identify key trends and patterns essential to predict the output.
- **Testing Data:** After the model is trained, it must be tested to evaluate how accurately it can predict an outcome. This is done by the testing data set.

Note: A Machine Learning process begins by feeding the machine lots of data, by using this data the machine is trained to detect hidden insights and trends. These insights are then used to build a Machine Learning Model by using an algorithm in order to solve a problem in Figure 1.2.

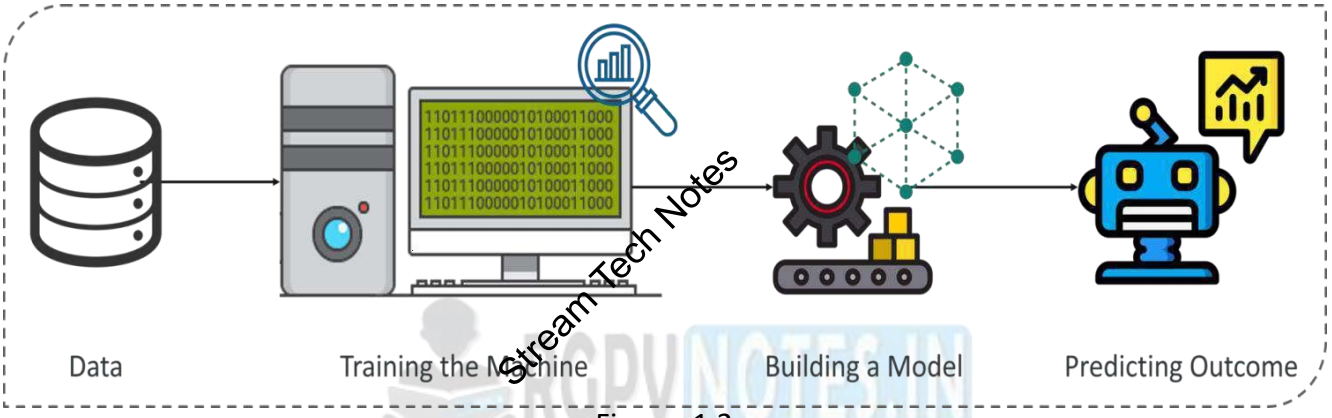


Figure: 1.2

Scope

- **Increase in Data Generation**: Due to excessive production of data, need a method that can be used to structure, analyze and draw useful insights from data. This is where Machine Learning comes in. It uses data to solve problems and find solutions to the most complex tasks faced by organizations.
- **Improve Decision Making:** By making use of various algorithms, Machine Learning can be used to make better business decisions.

For example, Machine Learning is used to forecast sales, predict downfalls in the stock market, identify risks and anomalies, etc.

- **Uncover patterns & trends in data:** Finding hidden patterns and extracting key insights from data is the most essential part of Machine Learning. By building predictive models and using statistical techniques, Machine Learning allows you to dig beneath the surface and explore the data at a minute scale. Understanding data and extracting patterns manually will take days, whereas Machine Learning algorithms can perform such computations in less than a second.
- **Solve complex problems:** Building self-driving cars, Machine Learning can be used to solve the most complex problems.

Limitations

1. What algorithms exist for learning general target function from specific training examples?
2. In what setting will particular algorithm converge to the desired function, given sufficient training data?
3. Which algorithm performs best for which types of problems and representations?

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

4. How much training data is sufficient?
5. When and how can prior knowledge held by the learner guide the process of generalizing from examples?
6. What is the best way to reduce the learning task to one more function approximation problem?
7. Machine Learning Algorithms Require Massive Stores of Training Data.
8. Labeling Training Data Is a Tedious Process.
9. Machines Cannot Explain Themselves.

Machine Learning Types:

A Machine can learn to solve a problem by any one of the following three approaches.
These are the ways in which a machine can learn:

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning

Regression

Regression models are used to predict a continuous value. Predicting prices of a house given the features of house like size, price etc. is one of the common examples of Regression. It is a supervised technique.

Types of Regression

1. Simple Linear Regression
2. Polynomial Regression
3. Support Vector Regression
4. Decision Tree Regression
5. Random Forest Regression

Stream Tech Notes
RGPVNOTES.IN

Simple Linear Regression

This is one of the most common and interesting type of Regression technique. Here we predict a target variable Y based on the input variable X. A linear relationship should exist between target variable and predictor and so comes the name Linear Regression.
Consider predicting the salary of an employee based on his/her age. We can easily identify that there seems to be a correlation between employee's age and salary (more the age more is the salary). The hypothesis of linear regression is- Y= a + bX
Y represents salary, X is employee's age and a and b are the coefficients of equation. So in order to predict Y (salary) given X (age), we need to know the values of a and b (the model's coefficients).

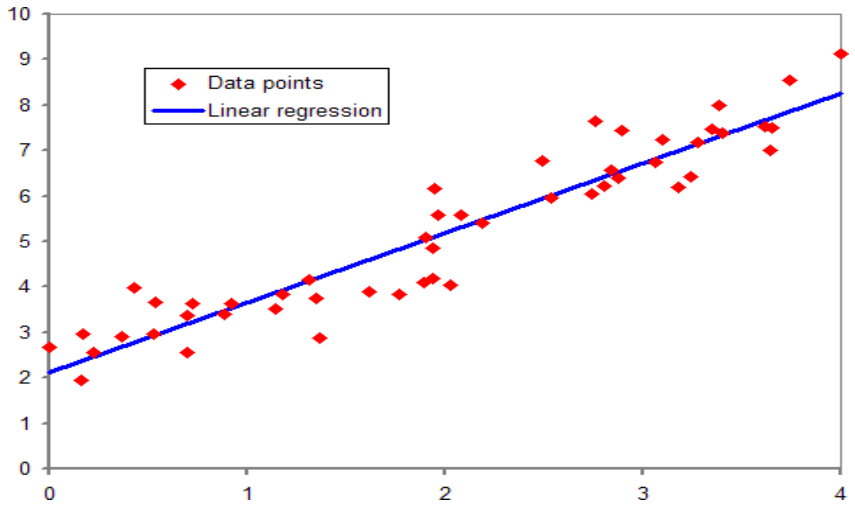


Figure: 1.3 Linear Regression

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

Polynomial Regression

In polynomial regression, we transform the original features into polynomial features of a given degree and then apply Linear Regression on it. Consider the above linear model Y = a+bX is transformed to something like – $Y=a + bX + cX^2$

It is still a linear model but the curve is now quadratic rather than a line. Scikit-Learn provide Polynomial Features class to transform the features.

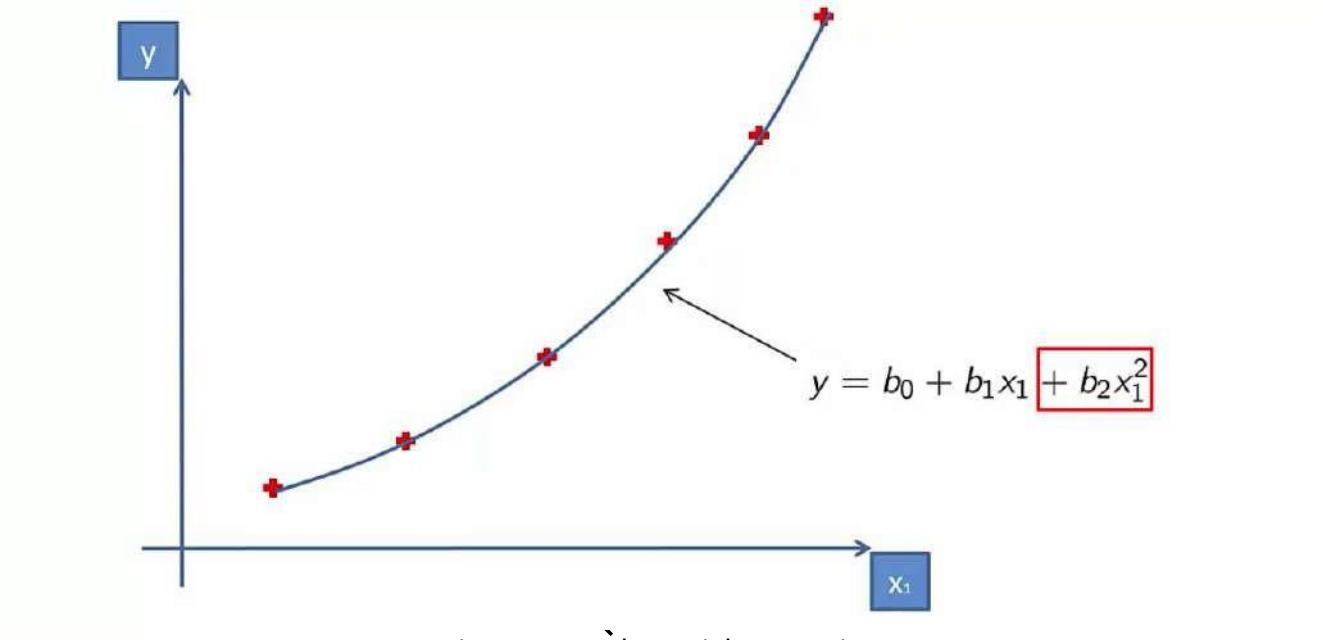


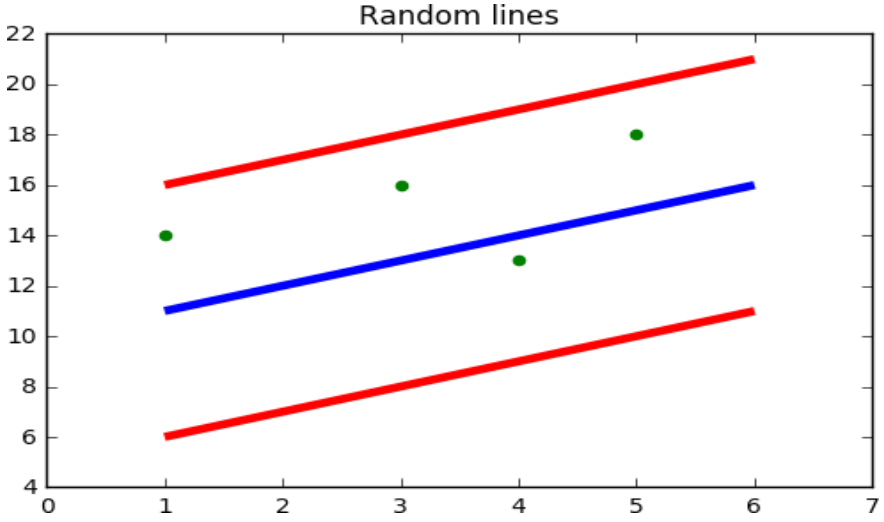
Figure: 1.4 Polynomial Regression

Stream

If we increase the degree to a very high value, the curve becomes overfitted as it learns the noise in the data as well.

Support Vector Regression

In SVR, we identify a hyper plane with maximum margin such that maximum numbers of data points are within that margin. SVRs are almost similar to SVM classification algorithm. Instead of minimizing the error rate as in simple linear regression, we try to fit the error within a certain threshold. Our objective in SVR is to basically consider the points that are within the margin. Our best fit line is the hyper plane that has maximum number of points.



Data points within the boundary line

Figure: 1.5 Support Vector Regression

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

Decision Tree Regression

Decision trees can be used for classification as well as regression. In decision trees, at each level we need to identify the splitting attribute. In case of regression, the ID3 algorithm can be used to identify the splitting node by reducing standard deviation.

A decision tree is built by partitioning the data into subsets containing instances with similar values (homogenous). Standard deviation is used to calculate the homogeneity of a numerical sample. If the numerical sample is completely homogeneous, its standard deviation is zero.

Random Forest Regression

Random forest is an ensemble approach where we take into account the predictions of several decision regression trees.

1. Select K random points
2. Identify n where n is the number of decision tree regressors to be created. Repeat step 1 and 2 to create several regression trees.
3. The average of each branch is assigned to leaf node in each decision tree.
4. To predict output for a variable, the average of all the predictions of all decision trees are taken into consideration.

Random Forest prevents over fitting (which is common in decision trees) by creating random subsets of the features and building smaller trees using these subsets.

Probability

probability is an intuitive concept. We use it on a daily basis without necessarily realising that we are speaking and applying probability to work.

Stream Tech Notes

Life is full of uncertainties. We don't know the outcomes of a particular situation until it happens. Will it rain today? Will I pass the next math test? Will my favorite team win the toss? Will I get a promotion in next 6 months? All these questions are examples of uncertain situations we live in. Let us map them to few common terminologies are-

- Experiment – are the uncertain situations, which could have multiple outcomes. Whether it rains on a daily basis is an experiment.
- Outcome is the result of a single trial. So, if it rains today, the outcome of today's trial from the experiment is "It rained"
- Event is one or more outcome from an experiment. "It rained" is one of the possible event for this experiment.
- Probability is a measure of how likely an event is. So, if it is 60% chance that it will rain tomorrow, the probability of Outcome "it rained" for tomorrow is 0.6

Random Variables

To calculate the likelihood of occurence of an event, we need to put a framework to express the outcome in numbers. We can do this by mapping the outcome of an experiment to numbers.

Let's define X to be the outcome of a coin toss.

X = outcome of a coin toss

Possible Outcomes:

- 1 if heads
- 0 if tails

Let's take another one.

Suppose, I win the game if I get a sum of 8 while rolling two fair dice. I can define my random variable Y to be (the sum of the upward face of two fair dice)

Y can take values = (2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)

A few things to note about random variables:

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

- Each value of the random variable may or may not be equally likely. There is only 1 combination of dice, with sum 2{(1,1)}, while a sum of 5 can be achieved by {(1,4), (2,3), (3,2), (4,1)}. So, 5 is more likely to occur as compared to 2. On the contrary, the likelihood of a head or a tail in a coin toss is equal and 50-50.
- Sometimes, the random variables can only take fixed values, or values only in a certain interval. For example in a dice, the top face will only show values between 1 and 6. It cannot take a 2.25 or a 1.5. Similarly, when a coin is flipped, it can only show heads and tails and nothing else. On the other hand, if I define my random variable to be the amount of sugar in orange. It can take any value like 1.4g, 1.45g, 1.456g, 1.4568g as so on. All these values are possible and all infinite values between them are also possible. So, in this case, the random variable is continuous with a possibility of all real numbers.
- Don't think random variable as a traditional variable (even though both are called variables) like y=x+2, where the value of y is dependent on x. Random variable is defined in terms of the outcome of a process. We quantify the process using the random variable.

Statistic

Machine learning and statistics are two tightly related fields of study. So much so that statisticians refer to machine learning as "applied statistics" or "statistical learning" rather than the computer-science-centric name.

Raw observations alone are data, but they are not information or knowledge.

Data raises questions, such as:

- What is the most common or expected observation?
- What are the limits on the observations?
- What does the data look like?

Stream Tech Notes

Although they appear simple, these questions must be answered in order to turn raw observations into information that we can use and share.

Beyond raw data, we may design experiments in order to collect observations. From these experimental results we may have more sophisticated questions, such as:

- What variables are most relevant?
- What is the difference in an outcome between two experiments?
- Are the differences real or the result of noise in the data?

Questions of this type are important. The results matter to the project, to stakeholders, and to effective decision making.

Statistical methods are required to find answers to the questions that we have about data.

We can see that in order to both understand the data used to train a machine learning model and to interpret the results of testing different machine learning models, that statistical methods are required. ***Statistics is a subfield of mathematics.***

It refers to a collection of methods for working with data and using data to answer questions.

Descriptive Statistics

Descriptive statistics refer to methods for summarizing raw observations into information that we can understand and share.

Commonly, we think of descriptive statistics as the calculation of statistical values on samples of data in order to summarize properties of the sample of data, such as the common expected value (e.g. the mean or median) and the spread of the data (e.g. the variance or standard deviation).

Descriptive statistics may also cover graphical methods that can be used to visualize samples of data. Charts and graphics can provide a useful qualitative understanding of both the shape or distribution of observations as well as how variables may relate to each other.

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

Inferential Statistics

Inferential statistics is a fancy name for methods that aid in quantifying properties of the domain or population from a smaller set of obtained observations called a sample.

Commonly, we think of inferential statistics as the estimation of quantities from the population distribution, such as the expected value or the amount of spread.

More sophisticated statistical inference tools can be used to quantify the likelihood of observing data samples given an assumption. These are often referred to as tools for statistical hypothesis testing, where the base assumption of a test is called the null hypothesis.

Linear Algebra

Linear Algebra is a branch of mathematics that lets you concisely describe coordinates and interactions of planes in higher dimensions and perform operations on them and concerned with vectors, matrices, and linear transforms.

Although linear algebra is integral to the field of machine learning, the tight relationship is often left unexplained or explained using abstract concepts such as vector spaces or specific matrix operations.

Linear Algebra is required -

- When working with data, such as tabular datasets and images.
- When working with data preparation, such as one hot encoding and dimensionality reduction.
- The ingrained use of linear algebra notation and methods in sub-fields such as deep learning, natural language processing, and recommender systems.

Examples of linear algebra in machine learning-

1. Dataset and Data Files
2. Images and Photographs
3. Linear Regression
4. Regularization
5. Principal Component Analysis
6. Singular-Value Decomposition
7. Latent Semantic Analysis
8. Recommender Systems
9. Deep Learning

Stream Tech Notes
RGPVNOTES.IN

For instance-

Images and Photographs

1. Perhaps you are more used to working with images or photographs in computer vision applications.
2. Each image that you work with is itself a table structure with a width and height and one pixel value in each cell for black and white images or 3 pixel values in each cell for a color image.
3. A photo is yet another example of a matrix from linear algebra.
4. Operations on the image, such as cropping, scaling, shearing, and so on are all described using the notation and operations of linear algebra.

Linear Regression

1. Linear regression is an old method from statistics for describing the relationships between variables.
2. It is often used in machine learning for predicting numerical values in simpler regression problems.
3. There are many ways to describe and solve the linear regression problem, i.e. finding a set of coefficients that when multiplied by each of the input variables and added together results in the best prediction of the output variable.

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

Convex Optimization

Optimization is a big part of machine learning. It is the core of most popular methods, from least squares regression to artificial neural networks.

These methods useful in the core implementation of a machine learning algorithm. It is required to implement own algorithm tuning scheme to optimize the parameters of a model for some cost function.

A good example may be the case where we want to optimize the hyper-parameters of a blend of predictions from an ensemble of multiple child models.

Machine learning algorithms use optimization all the time. We minimize loss, or error, or maximize some kind of score functions. Gradient descent is the "hello world" optimization algorithm covered on probably any machine learning course. It is obvious in the case of regression, or classification models, but even with tasks such as clustering we are looking for a solution that *optimally* fits our data (e.g. k-means minimizes the within-cluster sum of squares). So if you want to understand how the machine learning algorithms do work, learning more about optimization helps. Moreover, if you need to do things like hyper parameter tuning, then you are also directly using optimization.

Data visualization

Data visualization is an important skill in applied statistics and machine learning.

Statistics does indeed focus on quantitative descriptions and estimations of data. Data visualization provides an important suite of tools for gaining a qualitative understanding.

This can be helpful when exploring and getting to know a dataset and can help with identifying patterns, corrupt data, outliers, and much more. With a little domain knowledge, data visualizations can be used to express and demonstrate key relationships in plots and charts that are more visceral to yourself and stakeholders than measures of association or significance.

Stream Tech Notes

There are five key plots that need to know well for basic data visualization. They are:

- Line Plot
- Bar Chart
- Histogram Plot
- Box and Whisker Plot
- Scatter Plot

With knowledge of these plots, you can quickly get a qualitative understanding of most data that you come across.

Line Plot

A line plot is generally used to present observations collected at regular intervals.

The x-axis represents the regular interval, such as time. The y-axis shows the observations, ordered by the x-axis and connected by a line.

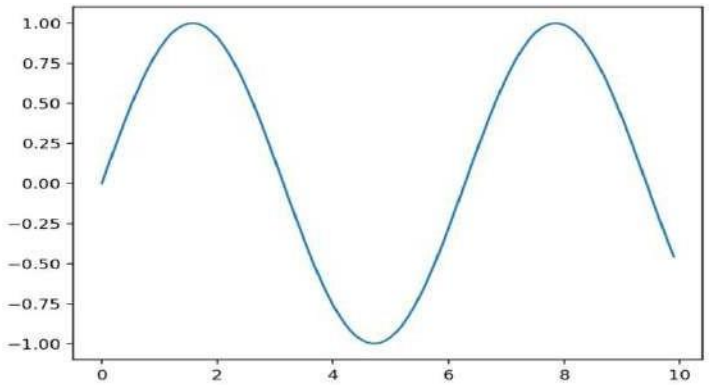


Figure: 1.6 Line Plot

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

Bar Chart

A bar chart is generally used to present relative quantities for multiple categories.

The x-axis represents the categories and are spaced evenly. The y-axis represents the quantity for each category and is drawn as a bar from the baseline to the appropriate level on the y-axis.

A bar chart can be created by calling the bar() function and passing the category names for the x-axis and the quantities for the y-axis.

Bar charts can be useful for comparing multiple point quantities or estimations.

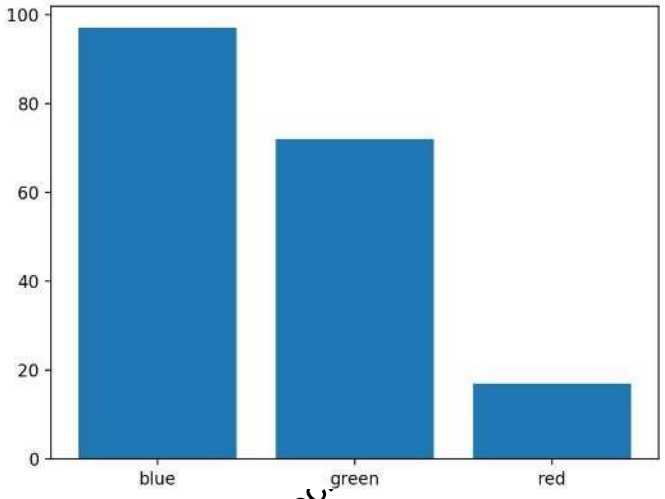


Figure: 1.7 Bar Chart

Stream Tec...

Histogram Plot

A histogram plot is generally used to summarize the distribution of a data sample.

The x-axis represents discrete bins or intervals for the observations. For example observations with values between 1 and 10 may be split into five bins, the values [1,2] would be allocated to the first bin, [3,4] would be allocated to the second bin, and so on.

The y-axis represents the frequency or count of the number of observations in the dataset that belong to each bin.

Essentially, a data sample is transformed into a bar chart where each category on the x-axis represents an interval of observation values.

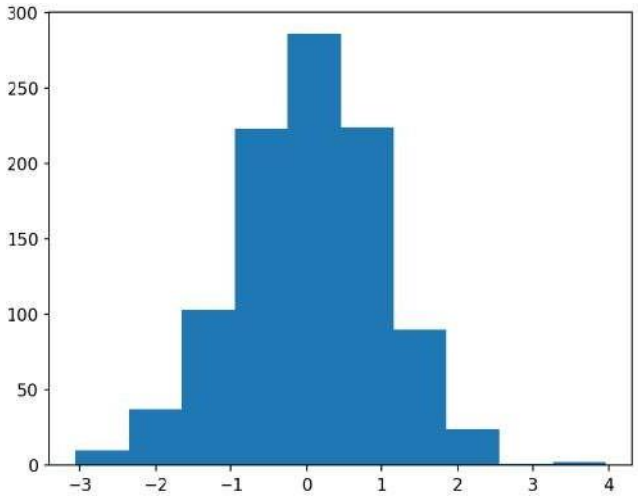


Figure: 1.7 Histogram Plot

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

Scatter Plot

A scatter plot (or 'scatterplot') is generally used to summarize the relationship between two paired data samples.

Paired data samples means that two measures were recorded for a given observation, such as the weight and height of a person.

The x-axis represents observation values for the first sample, and the y-axis represents the observation values for the second sample. Each point on the plot represents a single observation.

Scatter plots are useful for showing the association or correlation between two variables. A correlation can be quantified, such as a line of best fit, that too can be drawn as a line plot on the same chart, making the relationship clearer.

A dataset may have more than two measures (variables or columns) for a given observation. A scatter plot matrix is a cart containing scatter plots for each pair of variables in a dataset with more than two variables.

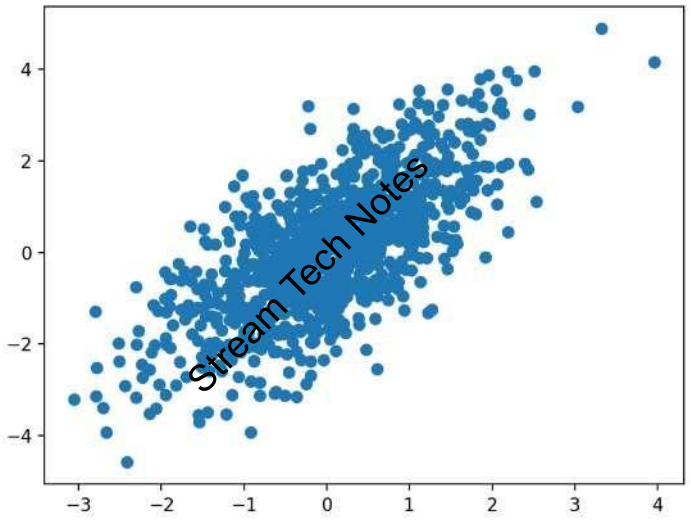


Figure: 1.7 Scatter Plot

Hypothesis function and testing

Hypothesis testing is a statistical method that is used in making statistical decisions using experimental data. Hypothesis Testing is basically an assumption that we make about the population parameter.

Ex : you say avg student in class is 40 or a boy is taller than girls.

all those example we assume need some statistic way to prove those. we need some mathematical conclusion what ever we are assuming is true.

Hypothesis testing is an essential procedure in statistics. A **hypothesis test** evaluates two mutually exclusive statements about a population to determine which statement is best supported by the sample data. When **we** say that a finding is statistically significant, it's thanks to a **hypothesis test**.

The process of hypothesis testing is to draw inferences or some conclusion about the overall population or data by conducting some statistical tests on a sample.

For drawing some inferences, we have to make some assumptions that lead to two terms that are used in the hypothesis testing.

- **Null hypothesis:** It is regarding the assumption that there is no anomaly pattern or believing according to the assumption made.
- **Alternate hypothesis:** Contrary to the null hypothesis, it shows that observation is the result of real effect.

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

Some of widely used hypothesis testing types :-

1. T Test (Student T test)
2. Z Test
3. ANOVA Test
4. Chi-Square Test

Data Distributions

A sample of data will form a distribution, and by far the most well-known distribution is the Gaussian distribution, often called the Normal distribution.

The distribution provides a parameterized mathematical function that can be used to calculate the probability for any individual observation from the sample space. This distribution describes the grouping or the density of the observations, called the probability density function. We can also calculate the likelihood of an observation having a value equal to or lesser than a given value. A summary of these relationships between observations is called a cumulative density function.

Distributions

From a practical perspective, we can think of a distribution as a function that describes the relationship between observations in a sample space.

For example, we may be interested in the age of humans, with individual ages representing observations in the domain, and ages 0 to 125 the extent of the sample space. The distribution is a mathematical function that describes the relationship of observations of different heights.

A distribution is simply a collection of data, or scores, on a variable. Usually, these scores are arranged in order from smallest to largest and then they can be presented graphically.

StreamTechNotes

Density Functions

Distributions are often described in terms of their density or density functions.

Density functions are functions that describe how the proportion of data or likelihood of the proportion of observations changes over the range of the distribution.

Two types of density functions are probability density functions and cumulative density functions.

- Probability Density function: calculates the probability of observing a given value.
- Cumulative Density function: calculates the probability of an observation equal or less than a value.

A probability density function, or PDF, can be used to calculate the likelihood of a given observation in a distribution. It can also be used to summarize the likelihood of observations across the distribution's sample space. Plots of the PDF show the familiar shape of a distribution, such as the bell-curve for the Gaussian distribution.

Data Pre-processing

Pre-processing refers to the transformations applied to our data before feeding it to the algorithm.

• Data pre-processing is a technique that is used to convert the raw data into a clean data set. In other words, whenever the data is gathered from different sources it is collected in raw format which is not feasible for the analysis.

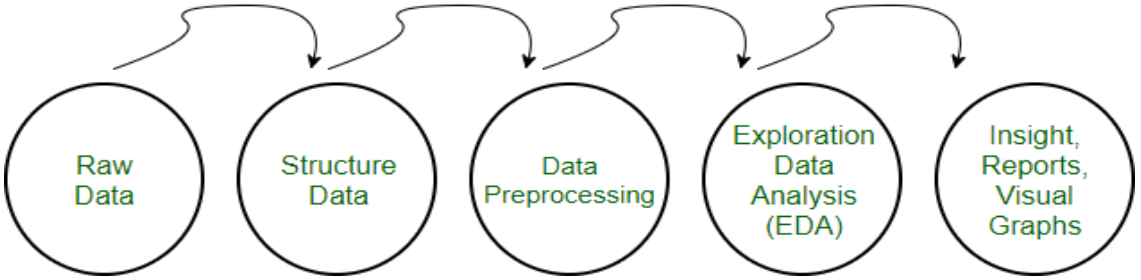


Figure: 1.8 Data Pre-Processing

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

Need of Data Pre-processing

• For achieving better results from the applied model in Machine Learning projects the format of the data has to be in a proper manner. Some specified Machine Learning model needs information in a specified format, for example, Random Forest algorithm does not support null values, therefore to execute random forest algorithm null values have to be managed from the original raw data set.
• Another aspect is that data set should be formatted in such a way that more than one Machine Learning and Deep Learning algorithms are executed in one data set, and best out of them is chosen.

Data Augmentation

Data augmentation is the process of increasing the amount and diversity of data. We do not collect new data, rather we transform the already present data. For instance we can consider image,so in image there are various ways to transform and augment the image data.

Need for data augmentation

Data augmentation is an integral process in deep learning, as in deep learning we need large amounts of data and in some cases it is not feasible to collect thousands or millions of images, so data augmentation comes to the rescue. It helps us to increase the size of the dataset and introduce variability in the dataset.

Operations in data augmentation

The most commonly used operations are-

1. Rotation
2. Shearing
3. Zooming
4. Cropping
5. Flipping
6. Changing the brightness level

Stream Tech Notes
RGPVNOTES.IN

Normalizing Data Sets

Normalization is a technique often applied as part of data preparation for machine learning. The goal of normalization is to change the values of numeric columns in the dataset to a common scale, without distorting differences in the ranges of values. For machine learning, every dataset does not require normalization. It is required only when features have different ranges.

The goal of normalization is to transform features to be on a similar scale. This improves the performance and training stability of the model.

Four common normalization techniques may be useful:

- scaling to a range
- clipping
- log scaling
- z-score

Normalization is a technique often applied as part of data preparation for machine learning. The goal of normalization is to change the values of numeric columns in the dataset to use a common scale, without distorting differences in the ranges of values or losing information. Normalization is also required for some algorithms to model the data correctly.

For example, assume your input dataset contains one column with values ranging from 0 to 1, and another column with values ranging from 10,000 to 100,000. The great difference in the scale of the numbers could cause problems when you attempt to combine the values as features during modelling. *Normalization* avoids these problems by creating new values that maintain the general distribution and ratios in the source data, while keeping values within a scale applied across all numeric columns used in the model.

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

Machine Learning Models

Types of classification algorithms in Machine Learning:

1. Linear Classifiers: Logistic Regression, Naive Bayes Classifier
2. Nearest Neighbour
3. Support Vector Machines
4. Decision Trees
5. Boosted Trees
6. Random Forest
7. Neural Networks

Naive Bayes Classifier (Generative Learning Model):

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability. Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Nearest Neighbour:

The k-nearest-neighbour algorithm is a classification algorithm, and it is supervised: it takes a bunch of labelled points and uses them to learn how to label other points. To label a new point, it looks at the labelled points closest to that new point (those are its nearest neighbours), and has those neighbour vote, so whichever label the most of the neighbours have is the label for the new point (the "k" is the number of neighbour it checks).

Smart Tech Notes

Logistic Regression (Predictive Learning Model):

It is a statistical method for analysing a data set in which there are one or more independent variables that determine an outcome. The outcome is measured with a dichotomous variable (in which there are only two possible outcomes). The goal of logistic regression is to find the best fitting model to describe the relationship between the dichotomous characteristic of interest (dependent variable = response or outcome variable) and a set of independent (predictor or explanatory) variables. This is better than other binary classification like nearest neighbour since it also explains quantitatively the factors that lead to classification.

Decision Trees:

Decision tree builds classification or regression models in the form of a tree structure. It breaks down a data set into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. A decision node has two or more branches and a leaf node represents a classification or decision. The topmost decision node in a tree which corresponds to the best predictor called root node. Decision trees can handle both categorical and numerical data.

Random Forest:

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of over fitting to their training set.

Neural Network:

A neural network consists of units (neurons), arranged in layers, which convert an input vector into some output. Each unit takes an input, applies a (often nonlinear) function to it and then passes the output on to the next layer. Generally the networks are defined to be feed-forward: a unit feeds its

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

output to all the units on the next layer, but there is no feedback to the previous layer. Weightings are applied to the signals passing from one unit to another, and it is these weightings which are tuned in the training phase to adapt a neural network to the particular problem at hand.

Types of Machine Learning

Machine learning is sub-categorized to three types:

1. **Supervised Learning – Train Me!**
2. **Unsupervised Learning – I am self-sufficient in learning**
3. **Reinforcement Learning – My life My rules! (Hit & Trial)**

Supervised Learning

Supervised Learning is the one, where you can consider the learning is guided by a teacher. We have a dataset which acts as a teacher and its role is to train the model or the machine. Once the model gets trained it can start making a prediction or decision when new data is given to it.

Figure 1.9 Supervised Learning

Unsupervised Learning

The model learns through observation and finds structures in the data. Once the model is given a dataset, it automatically finds patterns and relationships in the dataset by creating clusters in it. What it cannot do is add labels to the cluster; like it cannot say this a group of apples or mangoes, but it will separate all the apples from mangoes.

Suppose we presented images of apples, bananas and mangoes to the model, so what it does, based on some patterns and relationships it creates clusters and divides the dataset into those clusters. Now if a new data is fed to the model, it adds it to one of the created clusters.

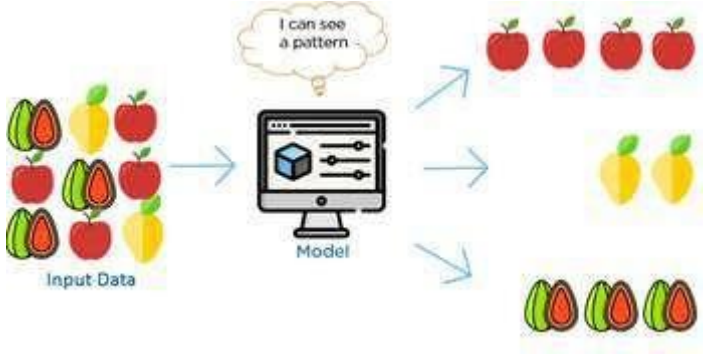


Figure 1.10 Un-Supervised Learning

Reinforcement Learning

It is the ability of an agent to interact with the environment and find out what is the best outcome. It

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

follows the concept of hit and trial method. The agent is rewarded or penalized with a point for a correct or a wrong answer, and on the basis of the positive reward points gained the model trains itself. And again once trained it gets ready to predict the new data presented to it.

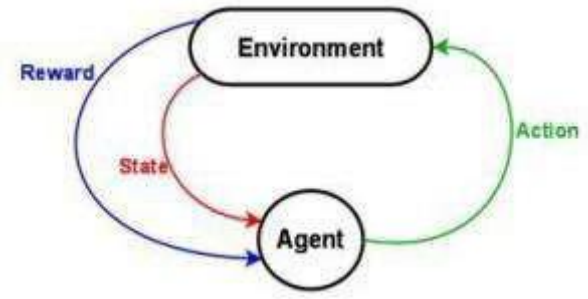
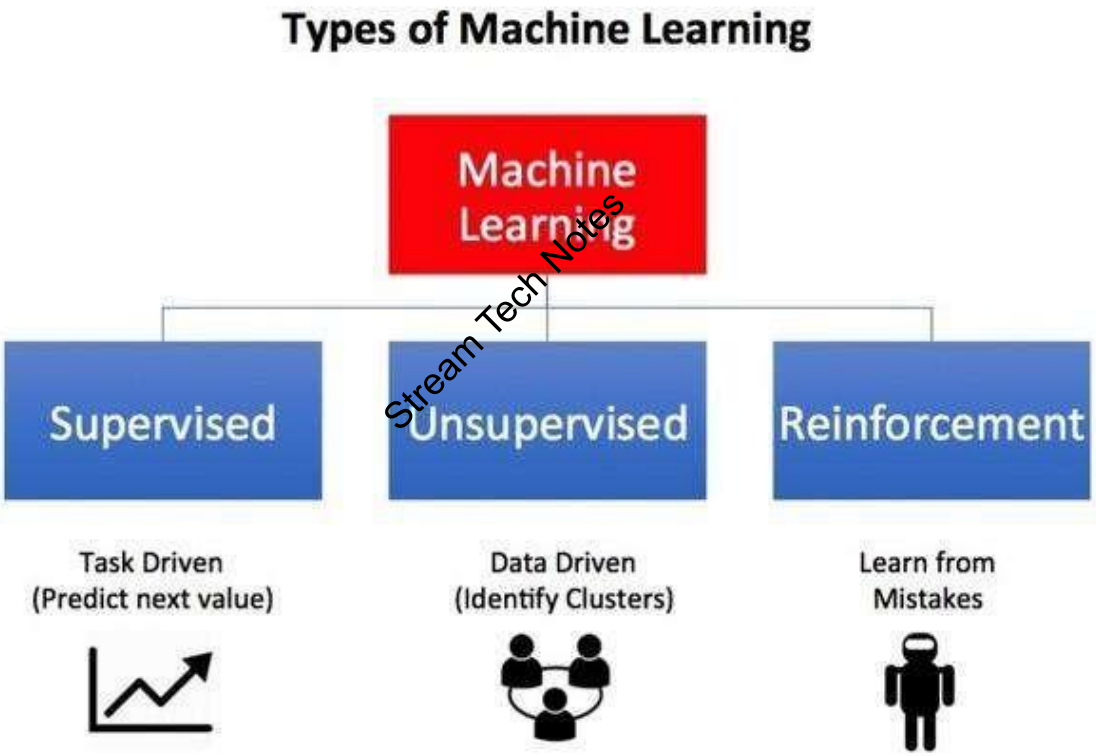


Figure 1.11 Un-Supervised Learning



Stream Tech Notes

Figure: 1.12 Types of Machine Learning

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

UNIT-2

Linearity vs non linearity

A linear model uses a linear function for its prediction function or as a crucial part of its prediction function.

A linear function takes a fixed number of numerical inputs, let's call them x1,x2,…,xn and returns

$$w_0 + \sum_{i=1}^{n} w_i x_i$$

where the weights w0,…,wn are the parameters of the model.

If the prediction function is a linear function, we can perform regression, i.e. predicting a numerical label. We can also take a linear function and return the sign of the result (whether the result is positive or not) and perform binary classification that way: all examples with a positive output receive label A, all others receive label B. There are various other (more complex) options for a response function on top of the linear function, the logistic function is very commonly used (which leads to logistic regression, predicting a number between 0 and 1, typically used to learn the probability of a binary outcome in a noisy setting).

A non-linear model is a model which is not a linear model. Typically these are more powerful (they can represent a larger class of functions) but much harder to train.

Activation functions like sigmoid,ReLU

A neural network is comprised of layers of nodes and learns to map examples of inputs to outputs.

For a given node, the inputs are multiplied by the weights in a node and summed together. This value is referred to as the summed activation of the node. The summed activation is then transformed via an activation function and defines the specific output or "activation" of the node. It is also known as Transfer Function.

StreamTech Notes

Activation function decides, whether a neuron should be activated or not by calculating weighted sum and further adding bias with it with the intention to introduce non-linearity into the output of a neuron.

It is used to determine the output of neural network like yes or no. It maps the resulting values in between 0 to 1 or -1 to 1 etc. (depending upon the function).

Activation functions are an extremely important feature of the artificial neural networks. They basically decide whether a neuron should be activated or not. Whether the information that the neuron is receiving is relevant for the given information or should it be ignored.

$$Y = \text{Activation}(\Sigma(weight * input) + bias)$$

The activation function is the non linear transformation that we do over the input signal. This transformed output is then sen to the next layer of neurons as input.

The Activation Functions can be basically divided into 2 types-

- Linear Activation Function
- Non-linear Activation Functions

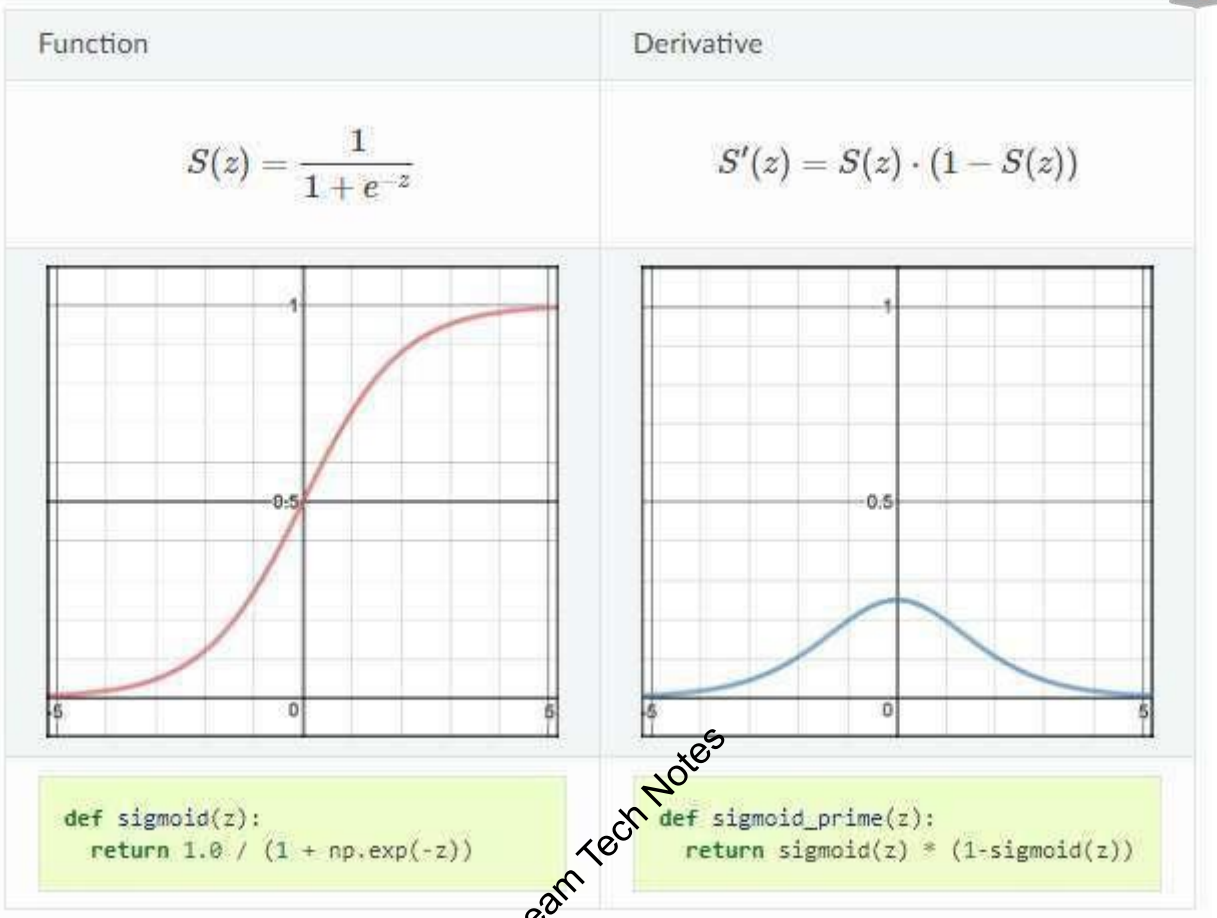
Sigmoid

Sigmoid takes a real value as input and outputs another value between 0 and 1. It's easy to work with and has all the nice properties of activation functions: it's non-linear, continuously differentiable, monotonic, and has a fixed output range.

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

RGPVNOTES.IN



Stream Tech Notes

Figure 2.1

Pros

- It is nonlinear in nature. Combinations of this function are also nonlinear!
- It will give an analog activation unlike step function.
- It has a smooth gradient too.
- It's good for a classifier.
- The output of the activation function is always going to be in range (0,1) compared to (-inf, inf) of linear function. So we have our activations bound in a range.

Cons

- Towards either end of the sigmoid function, the Y values tend to respond very less to changes in X.
- It gives rise to a problem of "vanishing gradients".
- 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.
- The network refuses to learn further or is drastically slow (depending on use case and until gradient /computation gets hit by floating point value limits).

ReLU

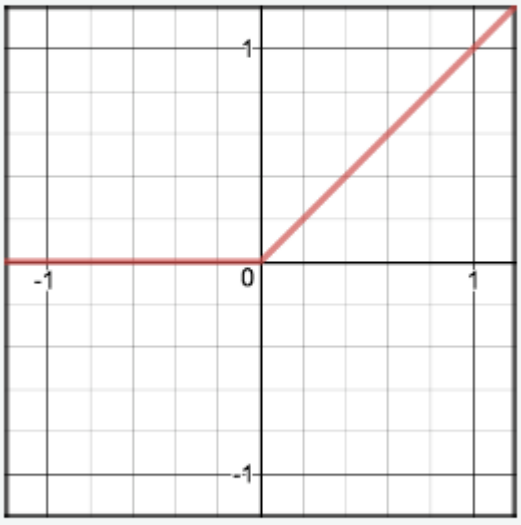
A recent invention which stands for Rectified Linear Units. The formula is deceptively simple: max(0,z). Despite its name and appearance, it's not linear and provides the same benefits as Sigmoid but with better performance.

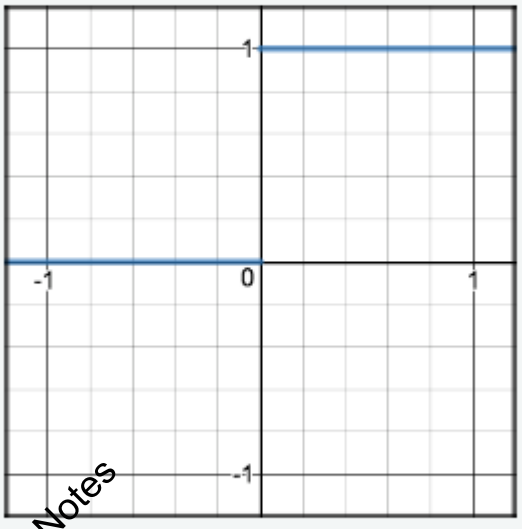
Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

RGPVNOTES.IN

Function	Derivative
$R(z) = \begin{cases} z & z > 0 \\ 0 & z <= 0 \end{cases}$	$R'(z) = \begin{cases} 1 & z > 0 \\ 0 & z < 0 \end{cases}$






```
def relu(z):
  return max(0, z)
```

```
def relu_prime(z):
  return 1 if z > 0 else 0
```

Stream Tech Notes

Figure 2.2

Weights and bias

Neural networks are a class of machine learning algorithms used to model complex patterns in datasets using multiple hidden layers and non-linear activation functions. A neural network takes an input, passes it through multiple layers of hidden neurons (mini-functions with unique coefficients that must be learned), and outputs a prediction representing the combined input of all the neurons.

$$Y = \sum (weight * input) + bias$$

Neuron(Node) — It is the basic unit of a neural network. It gets certain number of inputs and a bias value. When a signal(value) arrives, it gets multiplied by a weight value. If a neuron has 4 inputs, it has 4 weight values which can be adjusted during training time.

$$z = x_1 * w_1 + x_2 * w_2 + \ldots\ldots + x_n * w_n + b * 1$$

$$\hat{y} = a_{out} = sigmoid(z)$$

$$sigmoid(z) = \frac{1}{1 + e^{-z}}$$

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

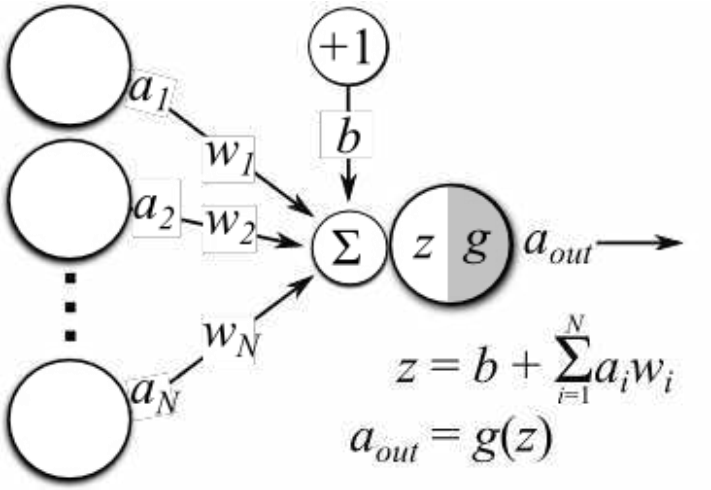


Figure 2.3 Operations at one neuron of a neural network

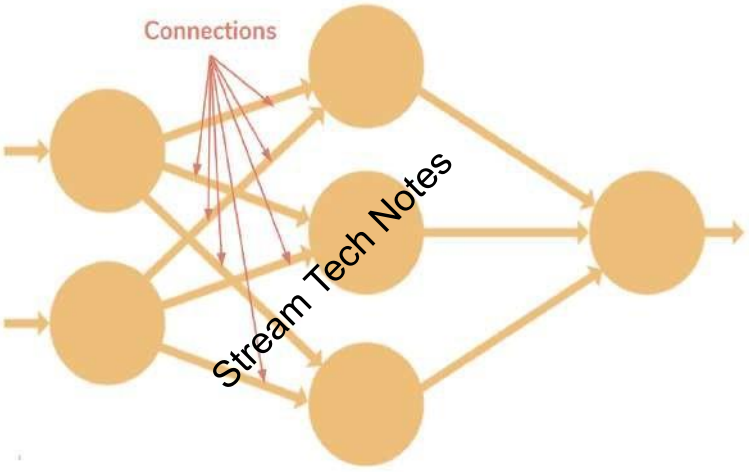


Figure 2.4

Connections — It connects one neuron in one layer to another neuron in other layer or the same layer. A connection always has a weight value associated with it. Goal of the training is to update this weight value to decrease the loss(error).

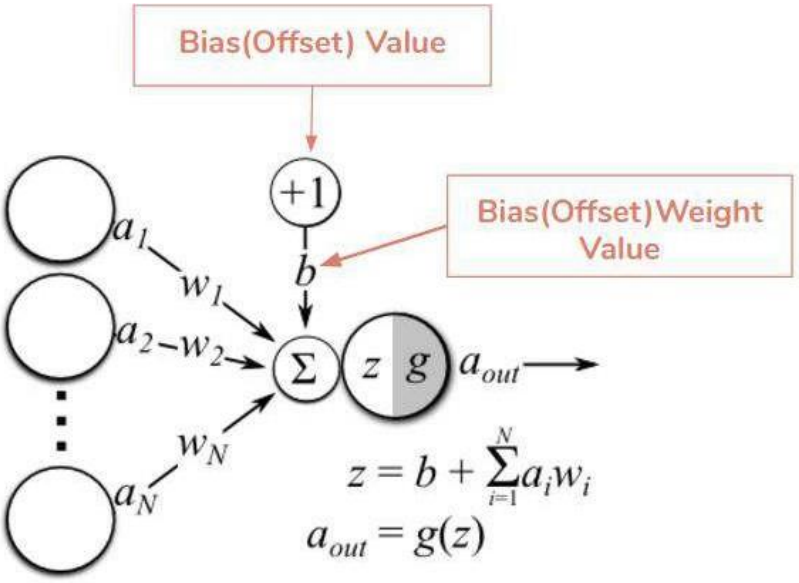


Figure 2.5

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

RGPVNOTES.IN

Bias (Offset) — It is an extra input to neurons and it is always 1, and has it's own connection weight. This makes sure that even when all the inputs are none (all 0's) there's gonna be an activation in the neuron. Bias terms are additional constants attached to neurons and added to the weighted input before the activation function is applied. Bias terms help models represent patterns that do not necessarily pass through the origin. For example, if all your features were 0, would your output also be zero? Is it possible there is some base value upon which your features have an effect? Bias terms typically accompany weights and must also be learned by your model.

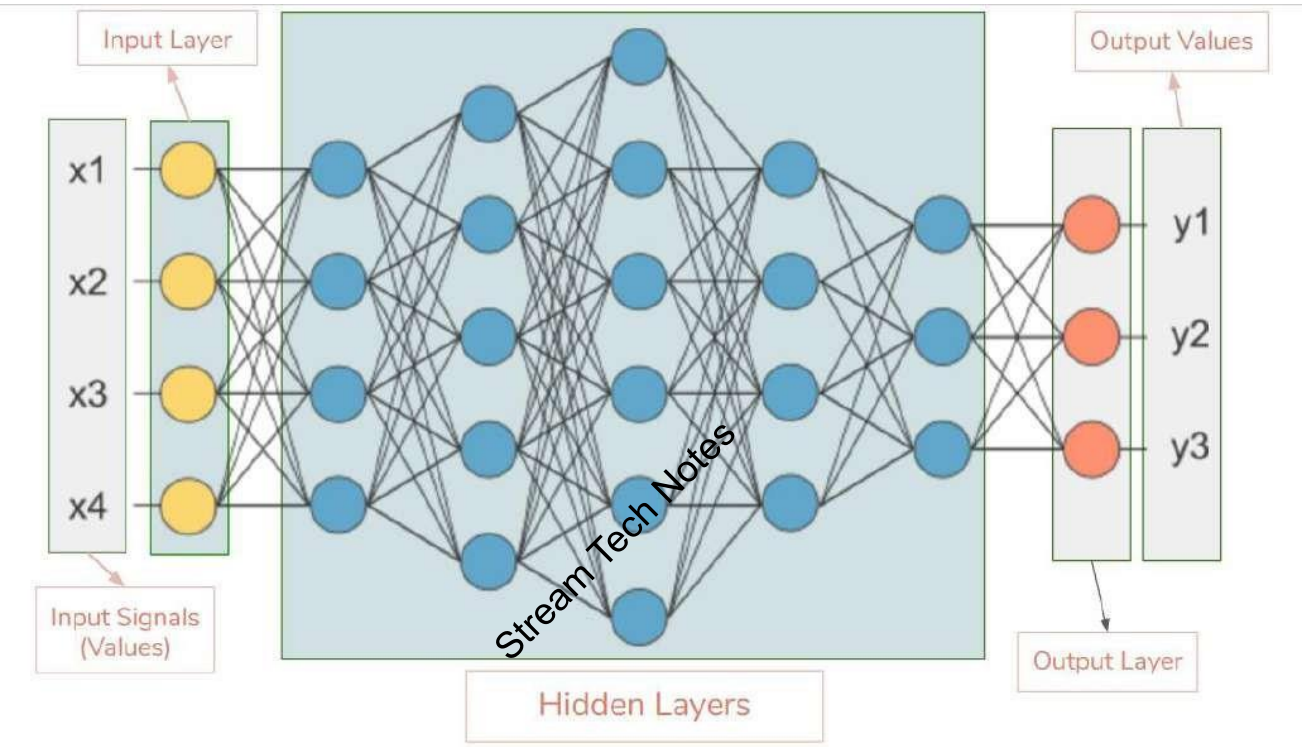


Figure 2.6

Input Layer — This is the first layer in the neural network. It takes input signals (values) and passes them on to the next layer. It doesn't apply any operations on the input signals (values) & has no weights and biases values associated. In our network we have 4 input signals x1, x2, x3, x4.

Hidden Layers — Hidden layers have neurons (nodes) which apply different transformations to the input data. One hidden layer is a collection of neurons stacked vertically (Representation). In our image given above we have 5 hidden layers. In our network, first hidden layer has 4 neurons (nodes), 2nd has 5 neurons, 3rd has 6 neurons, 4th has 4 and 5th has 3 neurons. Last hidden layer passes on values to the output layer. All the neurons in a hidden layer are connected to each and every neuron in the next layer, hence we have a fully connected hidden layers.

Output Layer — This layer is the last layer in the network & receives input from the last hidden layer. With this layer we can get desired number of values and in a desired range. In this network we have 3 neurons in the output layer and it outputs y1, y2, y3.

Input Shape — It is the shape of the input matrix we pass to the input layer. Our network's input layer has 4 neurons and it expects 4 values of 1 sample. Desired input shape for our network is (1, 4, 1) if we feed it one sample at a time. If we feed 100 samples input shape will be (100, 4, 1). Different libraries expect shapes in different formats.

Weights (Parameters) — A weight represent the strength of the connection between units. If the weight from node 1 to node 2 has greater magnitude, it means that neuron 1 has greater influence over neuron 2. A weight brings down the importance of the input value. Weights near zero means

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

changing this input will not change the output. Negative weights mean increasing this input will decrease the output. A weight decides how much influence the input will have on the output.
A neuron's input equals the sum of weighted outputs from all neurons in the previous layer. Each input is multiplied by the weight associated with the synapse connecting the input to the current neuron. If there are 3 inputs or neurons in the previous layer, each neuron in the current layer will have 3 distinct weights — one for each each synapse.

Single Input

$$Z = Input \cdot Weight = XW$$

$$Z = \sum_{i=1}^{n} x_i w_i = x_1 w_1 + x_2 w_2 + x_3 w_3$$

Multiple Input

StreamTech Notes

Notice, it's exactly the same equation we use with linear regression! In fact, a neural network with a single neuron is the same as linear regression! The only difference is the neural network post-processes the weighted input with an activation function.

Loss Functions

A loss function, or cost function, is a wrapper around our model's predict function that tells us "how good" the model is at making predictions for a given set of parameters. The loss function has its own curve and its own derivatives. The slope of this curve tells us how to change our parameters to make the model more accurate! We use the model to make predictions. We use the cost function to update our parameters. Our cost function can take a variety of forms as there are many different cost functions available. Popular loss functions include: MSE (L2) and Cross-entropy Loss.
The loss function computes the error for a single training example. The cost function is the average of the loss functions of the entire training set.

- '*mse*': for mean squared error.
- '*binary_crossentropy*': for binary logarithmic loss (logloss).
- '*categorical_crossentropy*': for multi-class logarithmic loss (logloss).

Gradient Descent

Optimization is a big part of machine learning. Almost every machine learning algorithm has an optimization algorithm at it's core.
Gradient descent is an optimization algorithm used to find the values of parameters (coefficients) of a function (f) that minimizes a cost function (cost).
Gradient descent is best used when the parameters cannot be calculated analytically (e.g. using linear algebra) and must be searched for by an optimization algorithm.

Gradient Descent Procedure

The procedure starts off with initial values for the coefficient or coefficients for the function. These could be 0.0 or a small random value.

coefficient = 0.0

The cost of the coefficients is evaluated by plugging them into the function and calculating the cost.

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

cost = f(coefficient)

or

cost = evaluate(f(coefficient))

The derivative of the cost is calculated. The derivative is a concept from calculus and refers to the slope of the function at a given point. We need to know the slope so that we know the direction (sign) to move the coefficient values in order to get a lower cost on the next iteration.

delta = derivative(cost)

Now that we know from the derivative which direction is downhill, we can now update the coefficient values. A learning rate parameter (alpha) must be specified that controls how much the coefficients can change on each update.

coefficient = coefficient – (alpha * delta)

This process is repeated until the cost of the coefficients (cost) is 0.0 or close enough to zero to be good enough.

Types of gradient Descent:

1. **Batch Gradient Descent:** This is a type of gradient descent which processes all the training examples for each iteration of gradient descent. But if the number of training examples is large, then batch gradient descent is computationally very expensive. Hence if the number of training examples is large, then batch gradient descent is not preferred. Instead, we prefer to use stochastic gradient descent or mini-batch gradient descent.
2. **Stochastic Gradient Descent:** This is a type of gradient descent which processes 1 training example per iteration. Hence, the parameters are being updated even after one iteration in which only a single example has been processed. Hence this is quite faster than batch gradient descent. But again, when the number of training examples is large, even then it processes only one example which can be additional overhead for the system as the number of iterations will be quite large.
3. **Mini Batch gradient descent:** This is a type of gradient descent which works faster than both batch gradient descent and stochastic gradient descent. Here *b* examples where *b<m* are processed per iteration. So even if the number of training examples is large, it is processed in batches of b training examples in one go. Thus, it works for larger training examples and that too with lesser number of iterations.

Stream Tech Notes

Multilayer network

A fully connected multi-layer neural network is called a Multilayer Perceptron (MLP).

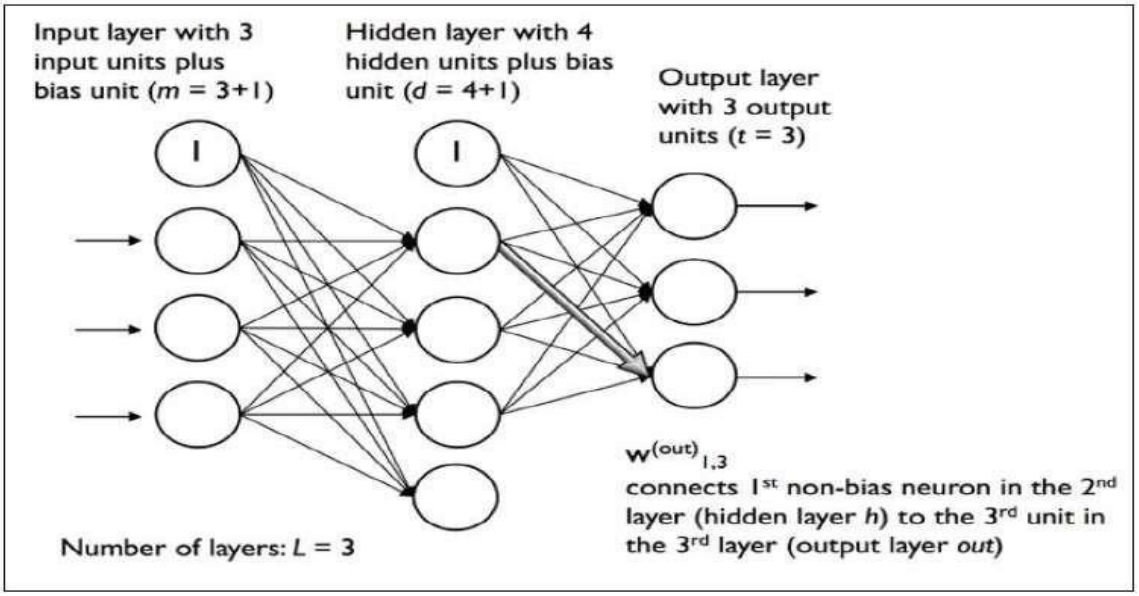


Figure 2.7

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

It has 3 layers including one hidden layer. If it has more than 1 hidden layer, it is called a deep ANN.
An MLP is a typical example of a feed-forward artificial neural network.
In this figure, the i^{th} activation unit in the l^{th} layer is denoted as $a_i^{(l)}$.
The number of layers and the number of neurons are referred to as hyper parameters of a neural network, and these need tuning. Cross-validation techniques must be used to find ideal values for these.
The weight adjustment training is done via backpropagation.

Notations

In the representation below:

$$a^{(in)} = \begin{bmatrix} a_0^{(in)} \\ a_1^{(in)} \\ \vdots \\ a_m^{(in)} \end{bmatrix} = \begin{bmatrix} 1 \\ x_1^{(in)} \\ \vdots \\ x_m^{(in)} \end{bmatrix}$$

Stream

- $a_i^{(in)}$ refers to the i^{th} value in the input layer
- $a_i^{(h)}$ refers to the i^{th} unit in the hidden layer
- $a_i^{(out)}$ refers to the i^{th} unit in the output layer
- $a_o^{(in)}$ is simply the bias unit and is equal to 1; it will have the corresponding weight w_0
- The weight coefficient from layer l to layer l+1 is represented by $w_{k,j}^{(l)}$

Backpropagation

Backpropagation is a supervised learning technique for neural networks that calculates the gradient of descent for weighting different variables. It's short for the backward propagation of errors, since the error is computed at the output and distributed backwards throughout the network's layers.
When an artificial neural network discovers an error, the algorithm calculates the gradient of the error function, adjusted by the network's various weights. The gradient for the final layer of weights is calculated first, with the first layer's gradient of weights calculated last. Partial calculations of the gradient from one layer are reused to determine the gradient for the previous layer. This point of this backwards method of error checking is to more efficiently calculate the gradient at each layer than the traditional approach of calculating each layer's gradient separately.

Uses of Backpropagation

Backpropagation is especially useful for deep neural networks working on error-prone projects, such as image or speech recognition. Taking advantage of the chain and power rules allows backpropagation to function with any number of outputs and better train all sorts of neural networks.

Unstable gradient problem

The unstable gradient problem is a fundamental problem that occurs in a neural network, that entails that a gradient in a deep neural network tends to either explode or vanish in early layers. The unstable gradient problem is not necessarily the vanishing gradient problem or the exploding

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

gradient problem, but is rather due to the fact that gradient in early layers is the product of terms from all proceeding layers. More layers make the network an intrinsically unstable solution. Balancing all products of terms is the only way each layer in a neural network can close at the same speed and avoid vanishing or exploding gradients. Balanced product of terms occurring by chance becomes more and more unlikely with more layers. Neural networks there for have layers that learn at different speeds, without being given any mechanisms or underlying reason for balancing learning speeds. When magnitudes of gradients accumulate, unstable networks are more likely to occur, which is a cause of poor prediction results.

Consider $y = f(x)$ that maps an input x to some output y. The function $f(\cdot)$ could be anything but here we consider a normal feed forward neural network.

Feedforward NN's have weight matrices and (non-linear) activation functions (that enable a non-linear decision surface, see also the XOR problem). In the elaboration below, let's look at the simplest imaginable "neural" network, with a single input (scalar x) and a single output (scalar y). Input passes through several layers (let's take 3 in the example), in each of which it's multiplied with a weight matrix W_l (l for layer index) and the resulting vector is fed through an activation function (let's take a sigmoid, σ). In our example, the input is a scalar and our weights are scalars as well. So the expression for $f(x)$ is:

$$f(x) = \sigma(w_3 \times \sigma(w_2 \times \sigma(w_1 \times x)))$$

Now let's train this w.r.t. some cost E. Whichever E we choose, we want the $\frac{\partial E}{\partial w_l}$ that constitute our gradient of E w.r.t. weights in order to find weights that minimize E. The choice of cost function doesn't matter for our purposes since $\frac{\partial E}{\partial w_l} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial w_l}$. So let's differentiate our expression for $f(x)$ w.r.t. the weights:

Stream Tech Notes

$$\frac{\partial f}{\partial w_1} = \sigma'(h_2 w_3) \times w_3 \times \sigma'(h_1 w_2) \times w_2 \times \sigma'(x w_1) x$$

Where $h_2 = \sigma(w_2 \times \sigma(w_1 \times x))$ and $h_1 = \sigma(w_1 \times x)$.

The expression for the partial derivative of E w.r.t. a weight in the first layer of (w_1) shows a lot of product terms. This product chain can be a source of gradient instabilities.

First of all, the product of $\sigma'(\cdot)$ exponentially decreases when going deeper. The function $\sigma'(\cdot) = \sigma(\cdot) \times (1 - \sigma(\cdot))$ has a peak value of 0.25 at 0. Thus, with a deep FF NN, in the k-th layer, the partial derivative of E for each weight in that layer will have a $(1/4)^k$ contribution from the sigmoid. With three layers, this is 0.0156, compared to the 0.25 at the top layer. This is assumed to be the cause of the vanishing gradient problem.

On the other hand, when weights in each layer are large, we could get an exploding gradient (because of the product of many large numbers). To conclude, the chain of products in the derivative for deeper layers causes gradient instabilities.

Auto encoders

An auto encoder **neural network** is an **Unsupervised Machine learning** algorithm that applies backpropagation, setting the target values to be equal to the inputs. Auto encoders are used to reduce the size of our inputs into a smaller representation. If anyone needs the original data, they can reconstruct it from the compressed data.

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

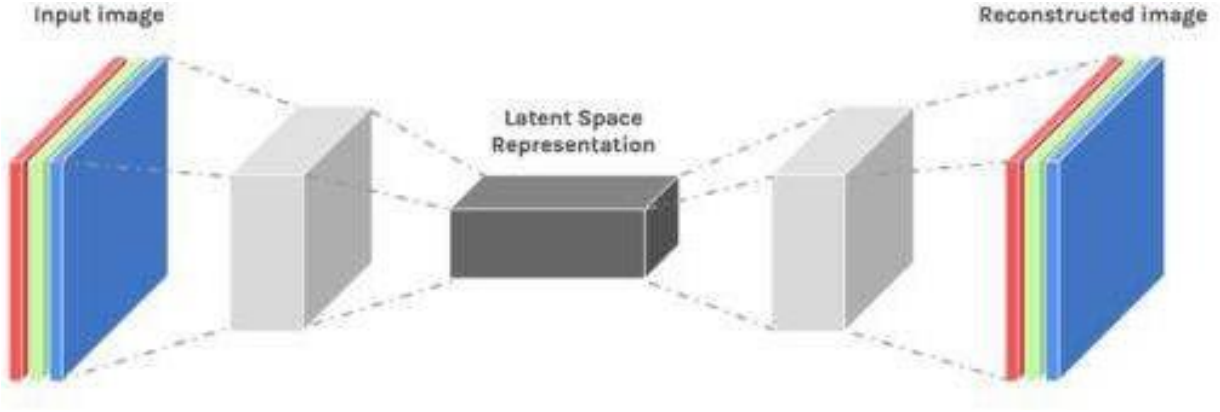
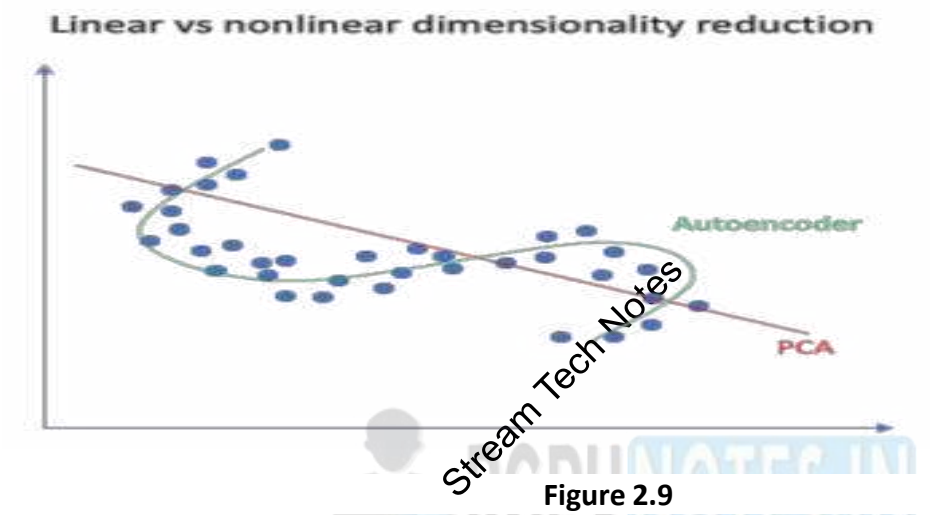


Figure 2.8



Stream Tech Notes

Figure 2.9

Auto encoders are preferred over PCA because:

- An auto encoder can learn **non-linear transformations** with a **non-linear activation function** and multiple layers.
- It doesn't have to learn dense layers. It can use **convolutional layers** to learn which is better for video, image and series data.
- It is more efficient to learn several layers with an auto encoder rather than learn one huge transformation with PCA.
- An auto encoder provides a representation of each layer as the output.
- It can make use of **pre-trained layers** from another model to apply transfer learning to enhance the encoder/decoder.

Applications of Auto encoders

1. **Image Coloring**

Auto encoders are used for converting any black and white picture into a colored image. Depending on what is in the picture, it is possible to tell what the color should be.

2. **Feature variation**

It extracts only the required features of an image and generates the output by removing any noise or unnecessary interruption.

3. **Dimensionality Reduction**

The reconstructed image is the same as our input but with reduced dimensions. It helps in providing the similar image with a reduced pixel value.

4. **Denoising Image**

The input seen by the auto encoder is not the raw input but a stochastically corrupted version. A denoising auto encoder is thus trained to reconstruct the original input from the noisy version.

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

5. **Watermark Removal**

It is also used for removing watermarks from images or to remove any object while filming a video or a movie.

Architecture of Auto encoders : An Auto encoder consist of three layers:

1. **Encoder**
2. **Code**
3. **Decoder**

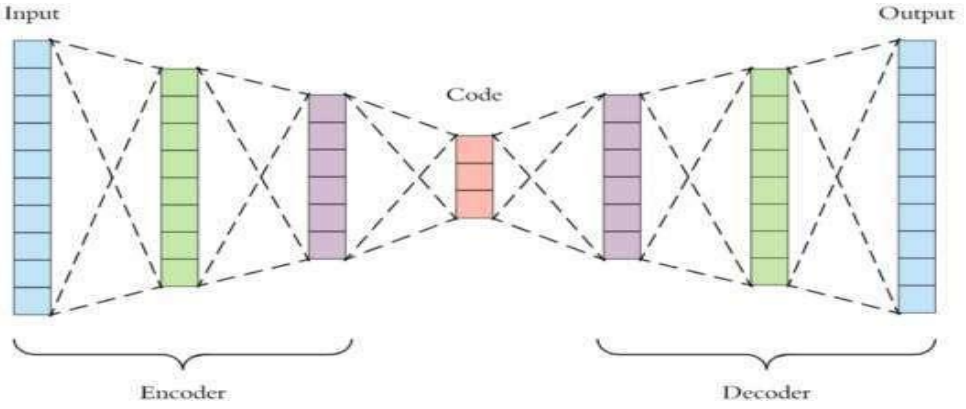


Figure 2.10

- **Encoder:** This part of the network compresses the input into a **latent space representation**. The encoder layer **encodes** the input image as a compressed representation in a reduced dimension. The compressed image is the distorted version of the original image.
- **Code:** This part of the network represents the compressed input which is fed to the decoder.
- **Decoder:** This layer **decodes** the encoded image back to the original dimension. The decoded image is a lossy reconstruction of the original image and it is reconstructed from the latent space representation.

Stream Tech Notes

The layer between the encoder and decoder, ie. the code is also known as **Bottleneck**. This is a well-designed approach to decide which aspects of observed data are relevant information and what aspects can be discarded. It does this by balancing two criteria :

- Compactness of representation, measured as the compressibility.
- It retains some behaviourally relevant variables from the input.

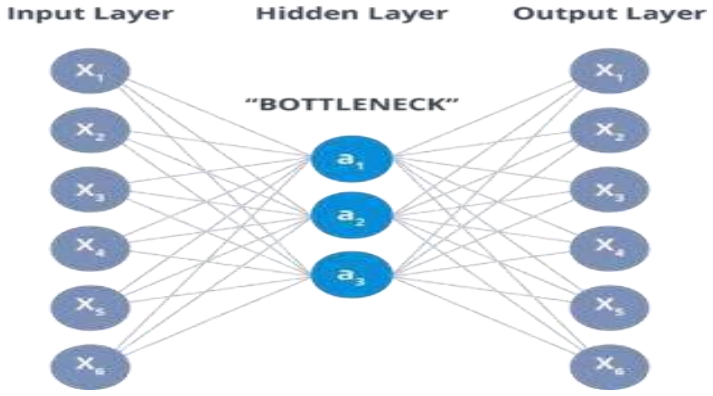


Figure 2.11

Types of Autoencoders

1. **Convolution Auto encoders**

Auto encoders in their traditional formulation does not take into account the fact that a signal can be seen as a sum of other signals. Convolutional Auto encoders use the convolution operator to exploit this observation. They learn to encode the input in a set of simple signals and then try to reconstruct

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

the input from them, modify the geometry or the reflectance of the image.

2. Sparse Auto encoders

Sparse auto encoders offer us an alternative method for introducing an information bottleneck **without requiring a reduction in the number of nodes** at our hidden layers. Instead, we'll construct our loss function such that we penalize activations within a layer.

3. Deep Auto encoders

The extension of the simple Auto encoder is the Deep Auto encoder. The first layer of the Deep Auto encoder is used for first-order features in the raw input. The second layer is used for second-order features corresponding to patterns in the appearance of first-order features. Deeper layers of the Deep Auto encoder tend to learn even higher-order features.

A **deep auto encoder** is composed of two, symmetrical deep-belief networks-

1. First four or five shallow layers representing the encoding half of the net.
2. The second set of four or five layers that make up the decoding half.

Batch normalization

Batch normalization is a method we can use to normalize the inputs of each layer, in order to fight the internal covariate shift problem.

During training time, a batch normalization layer does the following:

1. Calculate the mean and variance of the layers input.

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{Batch mean}$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \qquad \text{Batch variance}$$

Stream Tech Notes

Batch statistics for step 1

2. Normalize the layer inputs using the previously calculated batch statistics.

$$\overline{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

Normalization of the layers input in step 2

3. Scale and shift in order to obtain the output of the layer.

$$y_i = \gamma \overline{x_i} + \beta$$

Scaling and shifting the normalized input for step 3

Notice that *γ* and *β* are **learned during training** along with the original parameters of the network.

So, if each batch had *m* samples and there where *j* batches:

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

$$E_x = \frac{1}{m}\sum_{i=1}^{j}\mu_B^{(i)} \qquad \text{Inference mean}$$

$$Var_x = \left(\frac{m}{m-1}\right)\frac{1}{m}\sum_{i=1}^{j}\sigma_B^{2(i)} \qquad \text{Inference variance}$$

$$y = \frac{\gamma}{\sqrt{Var_x+\epsilon}}x + \left(\beta + \frac{\gamma E_x}{\sqrt{Var_x+\epsilon}}\right) \qquad \text{Inference scaling/shifting}$$

Inference formulas

Dropout

Dropout is implemented per-layer in a neural network.

It can be used with most types of layers, such as dense fully connected layers, convolutional layers, and recurrent layers such as the long short-term memory network layer.

Dropout may be implemented on any or all hidden layers in the network as well as the visible or input layer. It is not used on the output layer.

The term "dropout" refers to dropping out units (hidden and visible) in a neural network.

Simply , dropout refers to ignoring units (i.e. neurons) during the training phase of certain set of neurons which is chosen at random. By "ignoring", mean these units are not considered during a particular forward or backward pass.

More technically, At each training stage, individual nodes are either dropped out of the net with probability 1-p or kept with probability p, so that a reduced network is left; incoming and outgoing edges to a dropped-out node are also removed.

Stream Tech Notes

Neural networks are the building blocks of any machine-learning architecture. They consist of one input layer, one or more hidden layers, and an output layer.

When we training our neural network (or model) by updating each of its weights, it might become too dependent on the dataset we are using. Therefore, when this model has to make a prediction or classification, it will not give satisfactory results. This is known as over-fitting. We might understand this problem through a real-world example: If a student of mathematics learns only one chapter of a book and then takes a test on the whole syllabus, he will probably fail.

To overcome this problem, we use a technique that was introduced by Geoffrey Hinton in 2012. **This technique is known as dropout.**

L1 and L2 regularization

L1 and L2 regularisation owes its name to L1 and L2 norm of a vector ***w*** respectively. Here's a primer on norms:

$$||\mathbf{w}||_1 = |w_1| + |w_2| + ... + |w_N|$$

1-norm (also known as L1 norm)

$||\mathbf{w}||_2 = (w_1^2 + w_2^2 + ... + w_N^2)^{\frac{1}{2}}$

2-norm (also known as L2 norm or Euclidean norm)

$||\mathbf{w}||_p = (w_1^p + w_2^p + ... + w_N^p)^{\frac{1}{p}}$

p-norm

A linear regression model that implements L1 norm for regularisation is called **lasso regression**, and one that implements (squared) L2 norm for regularisation is called **ridge regression**. To implement these two, note that the linear regression model stays the same:

$$\hat{y} = w_1x_1 + w_2x_2 + ... + w_Nx_N + b$$

but it is the calculation of the loss function that includes these regularisation terms:

$$Loss = Error(y, \hat{y})$$

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

Loss function with no regularisation

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} |w_i|$$

Loss function with L1 regularisation

$$Loss = Error(y, \hat{y}) + \lambda \sum_{i=1}^{N} w_i^2$$

Loss function with L2 regularisation

The regularisation terms are 'constraints' by which an optimisation algorithm must 'adhere to' when minimising the loss function, apart from having to minimise the error between the true *y* and the predicted $\hat{y}$.

Loss Functions

To demonstrate the effect of L1 and L2 regularisation, let's fit our linear regression model using 3 different loss functions/objectives:

1. **L**
2. **L1**
3. **L2**

Our objective is to minimise these different losses.

1. **Loss function with no regularisation**

We define the loss function L as the squared error, where error is the difference between *y* (the true value) and $\hat{y}$ (the predicted value).

$$L = (\hat{y} - y)^2$$
$$= (wx + b - y)^2$$

Tech Notes

Let's assume our model will be overfitted using this loss function.

2. **Loss function with L1 regularisation**

Based on the above loss function, adding an L1 regularisation term to it looks like this:

$$L_1 = (wx + b - y)^2 + \lambda |w|$$

where the regularisation parameter $\lambda > 0$ is manually tuned. Let's call this loss function L1. Note that $|w|$ is differentiable everywhere except when w=0, as shown below. We will need this later.

$$\frac{d|w|}{dw} = \begin{cases} 1 & w > 0 \\ -1 & w < 0 \end{cases}$$

3. **Loss function with L2 regularisation**

Similarly, adding an L2 regularisation term to L looks like this:

$$L_2 = (wx + b - y)^2 + \lambda w^2$$

where again, $\lambda > 0$.

L2 loss function	L1 loss function
Not very robust	Robust
Stable solution	Unstable solution
Always one solution	Possibly multiple solutions

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

Momentum

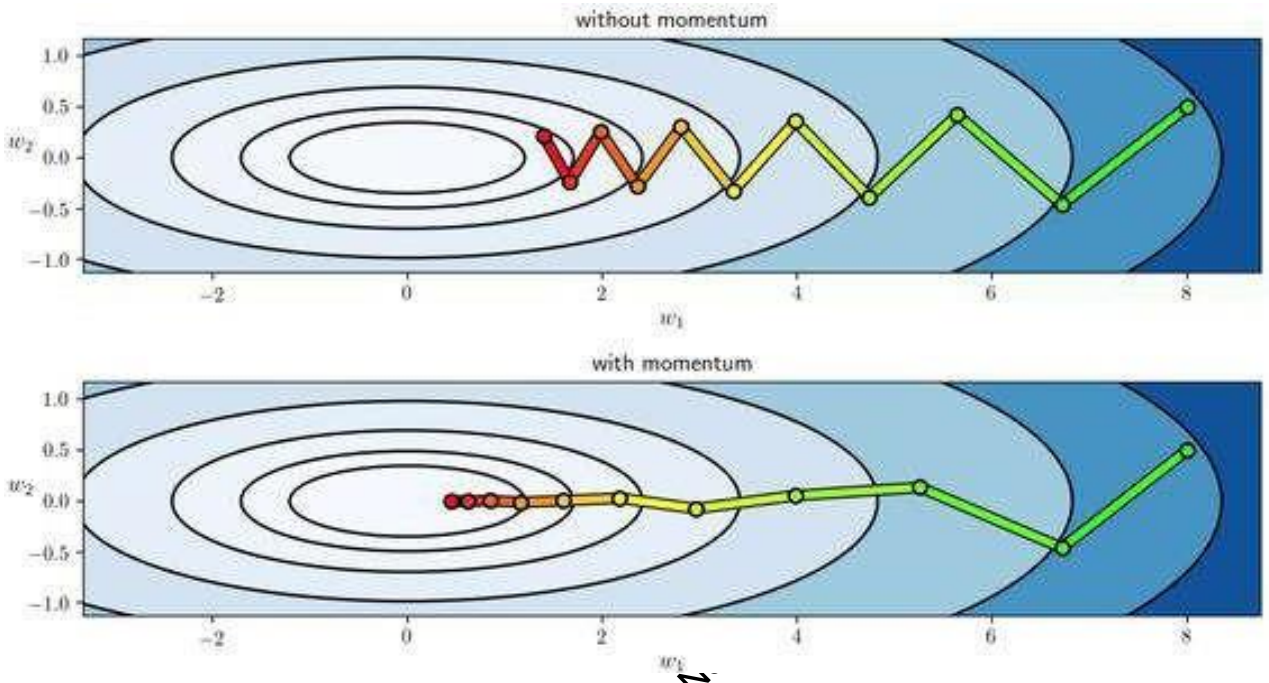


Figure 2.12

StreamTech N

Momentum methods in the context of machine learning refer to a group of tricks and techniques designed to speed up convergence of first-order optimization methods like gradient descent (and its many variants).

They essentially work by adding what's called *the momentum term* to the update formula for gradient descent, thereby ameliorating its natural "zigzagging behavior," especially in long narrow valleys of the cost function.

The figure below shows the progress of gradient descent - with and without momentum - towards reaching the minimum of a quadratic cost function, located at the center of the concentric elliptical contours.

Let's say your first update to the weights is a vector θ1θ1. For the second update (which would be θ2θ2 without momentum) you update by θ2+αθ1θ2+αθ1. For the next one, you update by θ3+αθ2+α2θ1θ3+αθ2+α2θ1, and so on. Here the parameter 0≤α<10≤α<1 indicates the amount of momentum we want.

The practical way of doing that is keeping an update vector vivi, and updating it as vi+1=αvi+θi+1vi+1=αvi+θi+1.

The reason we do this is to avoid the algorithm getting stuck in a local minimum. Think of it as a marble rolling around on a curved surface. We want to get to the lowest point. The marble having momentum will allow it to avoid a lot of small dips and make it more likely to find a better local solution.

Having momentum too high means you will be more likely to overshoot (the marble goes through the local minimum but the momentum carries it back upwards for a bit). This will lead to longer learning times. Finding the correct value of the momentum will depend on the particular problem: the smoothness of the function, how many local minima you expect, how "deep" the sub-optimal local minima are expected to be, etc.

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

RGPVNOTES.IN

Tuning hyper parameters

Hyper parameters, that cannot be directly learned from the regular training process. They are usually fixed before the actual training process begins. These parameters express important properties of the model such as its complexity or how fast it should learn.

Some examples of model hyper parameters include:

1. The penalty in Logistic Regression Classifier i.e. L1 or L2 regularization
2. The learning rate for training a neural network.
3. The C and sigma hyper parameters for support vector machines.
4. The k in k-nearest neighbors.

Models can have many hyper parameters and finding the best combination of parameters can be treated as a search problem. Two best strategies for Hyper parameter tuning are:

- Grid Search CV
- Randomized Search CV

Grid Search CV

In Grid Search CV approach, machine learning model is evaluated for a range of hyper parameter values. This approach is called Grid Search CV, because it searches for best set of hyper parameters from a grid of hyper parameters values.

For example, if we want to set two hyper parameters C and Alpha of Logistic Regression Classifier model, with different set of values. The grid search technique will construct many versions of the model with all possible combinations of hyper parameters, and will return the best one.

RGPV NOTES

C \ Alpha	0.1	0.2	0.3	0.4
0.5	0.701	0.703	0.697	0.696
0.4	0.699	0.702	0.698	0.702
0.3	0.721	0.726	0.713	0.703
0.2	0.706	0.705	0.704	0.701
0.1	0.698	0.692	0.688	0.675

As in the image, for C = [0.1, 0.2, 0.3, 0.4, 0.5] and Alpha = [0.1, 0.2, 0.3, 0.4].

For a combination ***C=0.3 and Alpha=0.2***, performance score comes out to be **0.726(Highest)**, therefore it is selected.

Drawback : Grid Search CV will go through all the intermediate combinations of hyper parameters which makes grid search computationally very expensive.

Randomized Search CV

Randomized Search CV solves the drawbacks of Grid Search CV, as it goes through only a fixed number of hyper parameter settings. It moves within the grid in random fashion to find the best set hyper parameters. This approach reduces unnecessary computation.

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

Unit –III

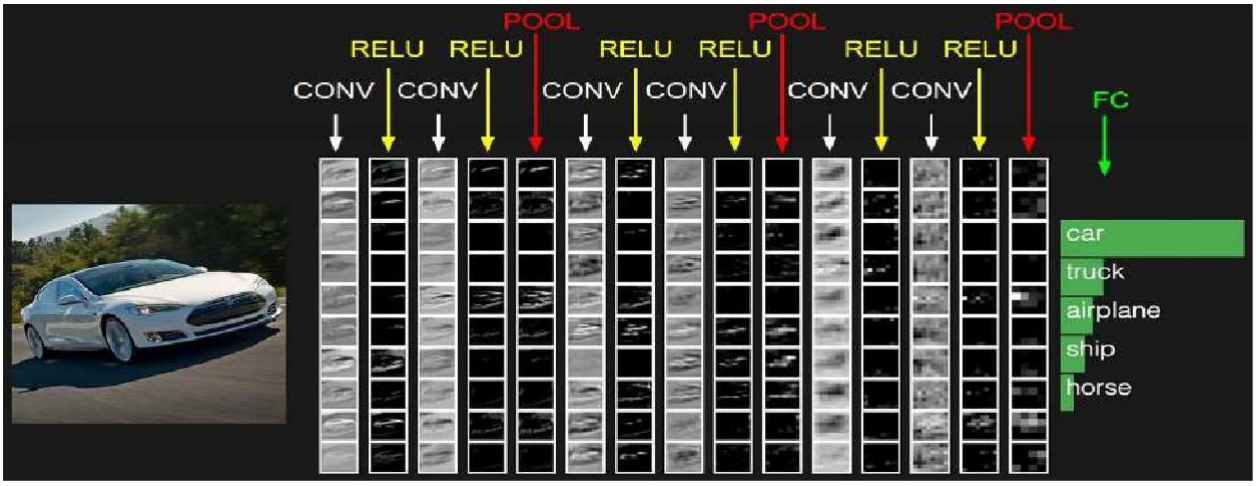
Convolutional neural network, flattening, subsampling, padding, stride, convolution layer, pooling layer, loss layer, dance layer 1x1 convolution, inception network, input channels, transfer learning, one shot learning, dimension reductions, implementation of CNN like tensor flow, keras etc

1. *Convolutional Neural Networks(CNNs/ConvNets)*

Convolutional Neural Networks are very similar to ordinary Neural Networks from the previous chapter: they are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they still have a loss function (e.g. SVM/Softmax) on the last (fully-connected) layer and all the tips/tricks we developed for learning regular Neural Networks still apply.

So what changes? ConvNet architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network. As we described above, a simple ConvNet is a sequence of layers, and every layer of a ConvNet transforms one volume of activations to another through a differentiable function. We use three main types of layers to build ConvNet architectures: **Convolutional Layer**, **Pooling Layer**, and **Fully-Connected Layer** (exactly as seen in regular Neural Networks). We will stack these layers to form a full ConvNet

Stream Tech Notes



Architecture.

Example Architecture: Overview. We will go into more details below, but a simple ConvNet for CIFAR-10 classification could have the architecture [INPUT - CONV - RELU - POOL - FC]. In more detail:

- INPUT [32x32x3] will hold the raw pixel values of the image, in this case an image of width 32, height 32, and with three color channels R,G,B.

Get real-time updates from RGPV

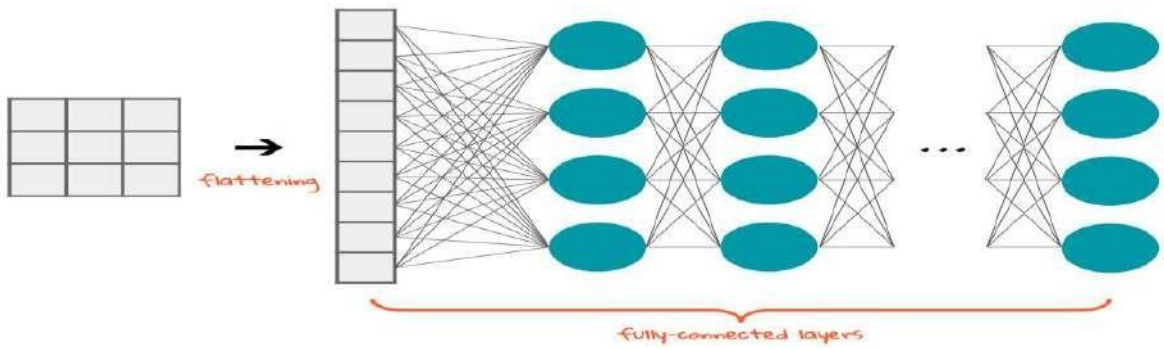
- CONV layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as [32x32x12] if we decided to use 12 filters.
- RELU layer will apply an element wise activation function, such as the max(0,x)max(0,x) thresholding at zero. This leaves the size of the volume unchanged ([32x32x12]).
- POOL layer will perform a downsampling operation along the spatial dimensions (width, height), resulting in volume such as [16x16x12].
- FC (i.e. fully-connected) layer will compute the class scores, resulting in volume of size [1x1x10], where each of the 10 numbers correspond to a class score, such as among the 10 categories of CIFAR-10. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.

In this way, ConvNets transform the original image layer by layer from the original pixel values to the final class scores. Note that some layers contain parameters and other don't. In particular, the CONV/FC layers perform transformations that are a function of not only the activations in the input volume, but also of the parameters (the weights and biases of the neurons). On the other hand, the RELU/POOL layers will implement a fixed function. The parameters in the CONV/FC layers will be trained with gradient descent so that the class scores that the ConvNet computes are consistent with the labels in the training set for each image.

StreamTech Notes

2. ***Flattening***

Flattening is converting the data into a 1-dimensional array for inputting it to the next layer. We flatten the output of the convolutional layers to create a single long feature vector. And it is connected to the final classification model, which is called a ***fully-connected*** layer. In other words, we put all the pixel data in one line and make connections with the final layer. And once again



3. ***Subsampling (*Supersample)**

Get real-time updates from RGPV

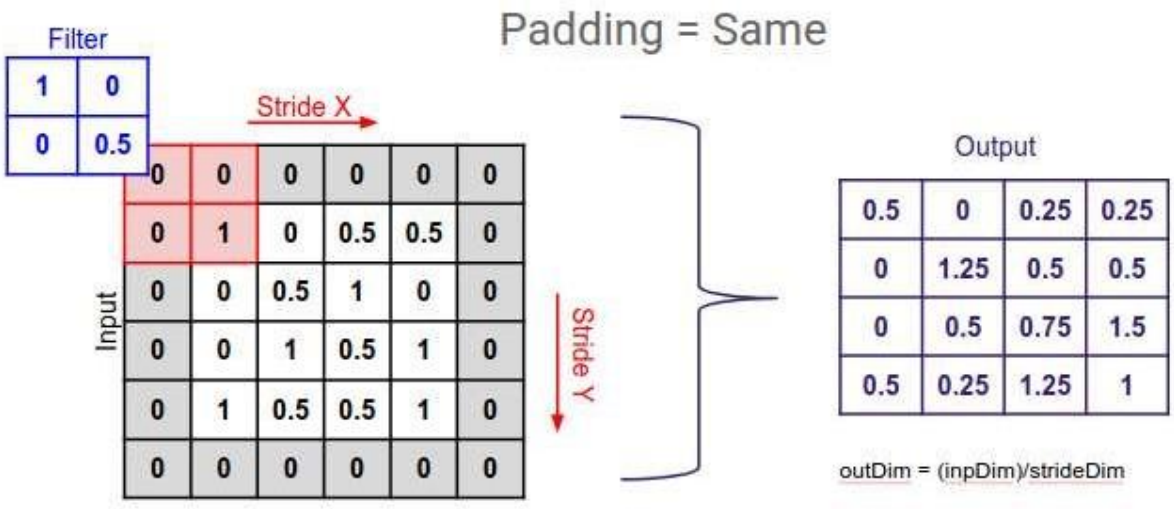
- Data Set is the entire collection of data to be analyzed. For inferential purposes, this may be treated as having been sampled from a population. All of the data set items will be classified by the process.
- Supersample is a subset of the data set chosen by simple random sampling. In our examples, it is the entire data set, but for larger data sets it will be considerably smaller. All computations prior to the final classification are performed on the supersample. For problems in moderate dimension (up to 50), the supersample will never need to be larger than 100,000–1,000,000 points, since the estimation error in a sample of this size is already too small to matter.
- Sample is one of several (RsRs) of size NsNs chosen by simple random sampling from the supersample. All intensive search operations are conducted in the sample so that the supersample is only used for one iteration from the best solution found in the sample. The sample size NsNs should be chosen to be large enough to reflect the essential structure of the data, while being small enough to keep the computations feasible.
- Subsample is one of several (RrRr) of size NrNr chosen by simple random sampling from the sample that is used to begin iterations on the sample. This number should be very small because great diversity in starting points generates diversity in solutions, and increases the chance of finding the best local maximum of the likelihood.

StreamTech Notes

4. What is Padding in Machine Learning?

Padding is a term relevant to convolutional neural networks as it refers to the amount of pixels added to an image when it is being processed by the kernel of a CNN. For example, if the padding in a CNN is set to zero, then every pixel value that is added will be of value zero. If, however, the zero padding is set to one, there will be a one pixel border added to the image with a pixel value of zero.

Get real-time updates from RGPV



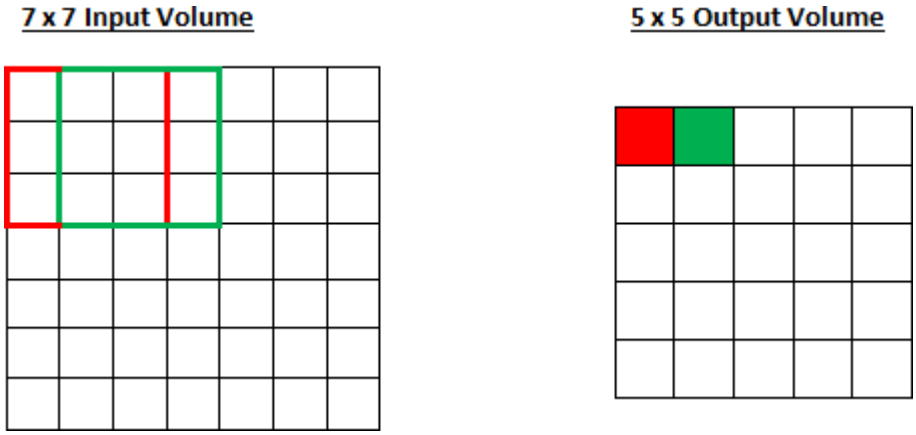
How does Padding work?

Padding works by extending the area of which a convolutional neural network processes an image. The kernel is the neural networks filter which moves across the image, scanning each pixel and converting the data into a smaller, or sometimes larger, format. In order to assist the kernel with processing the image, padding is added to the frame of the image to allow for more space for the kernel to cover the image. Adding padding to an image processed by a CNN allows for more accurate analysis of images.

Stream Tech Notes

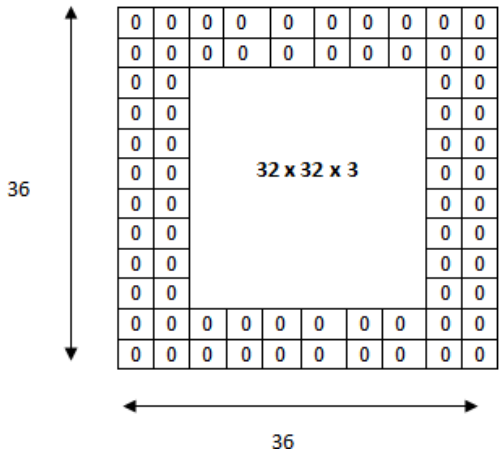
5. *What is Stride (Machine Learning)?*

Stride is a component of convolutional neural networks, or neural networks tuned for the compression of images and video data. Stride is a parameter of the neural network's filter that modifies the amount of movement over the image or video. For example, if a neural network's stride is set to 1, the filter will move one pixel, or unit, at a time. The size of the filter affects the encoded output volume, so stride is often set to a whole integer, rather than a fraction or decimal.



Get real-time updates from RGPV

Imagine a convolutional neural network is taking an image and analyzing the content. If the filter size is 3x3 pixels, the contained nine pixels will be converted down to 1 pixel in the output layer. Naturally, as the stride, or movement, is increased, the resulting output will be smaller. Stride is a parameter that works in conjunction with padding, the feature that adds blank, or empty pixels to the frame of the image to allow for a minimized reduction of size in the output layer. Roughly, it is a way of increasing the size of an image, to counteract the fact that stride reduces the size. Padding and stride are the foundational parameters of any convolutional neural network.



The input volume is 32 x 32 x 3. If we imagine two borders of zeros around the volume, this gives us a 36 x 36 x 3 volume. Then, when we apply our conv layer with our three 5 x 5 x 3 filters and a stride of 1, then we will also get a 32 x 32 x3 output volume.

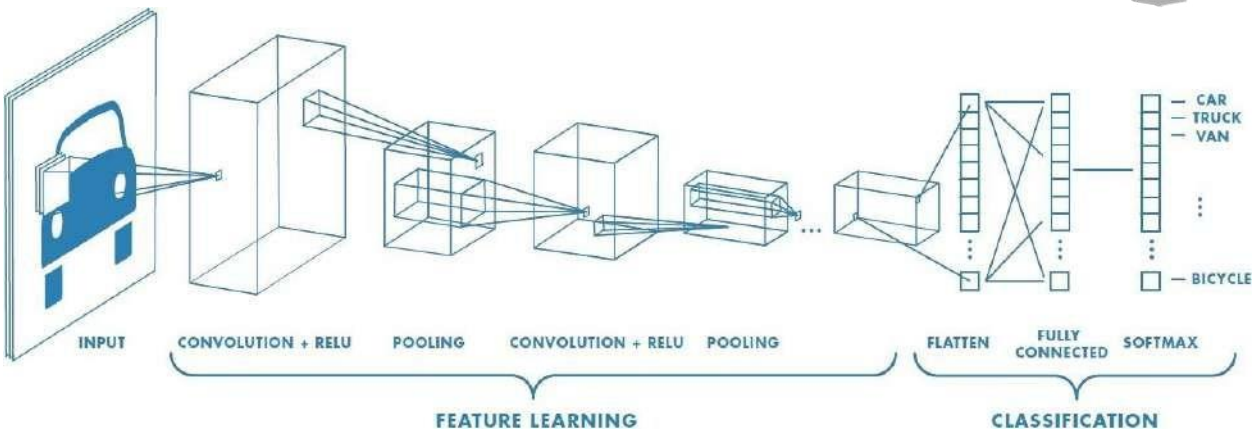
StreamTe-

In neural networks, Convolutional neural network (ConvNets or CNNs) is one of the main categories to do images recognition, images classifications. Objects detections, recognition faces etc., are some of the areas where CNNs are widely used.CNN image classifications takes an input image, process it and classify it under certain categories (Eg., Dog, Cat, Tiger, Lion). Computers sees an input image as array of pixels and it depends on the image resolution. Based on the image resolution, it will see h x w x d(h = Height, w = Width, d = Dimension). Eg., An image of 6 x 6 x 3 array of matrix of RGB (3 refers to RGB values) and an image of 4 x 4 x 1 array of matrix of grayscale image.

Technically, deep learning CNN models to train and test, each input image will pass it through a series of convolution layers with filters (Kernals), Pooling, fully connected layers (FC) and apply Softmax function to classify an object with probabilistic values between 0 and 1. The below figure is a complete flow of CNN to process an input image and classifies the objects based on values.

Get real-time updates from RGPV

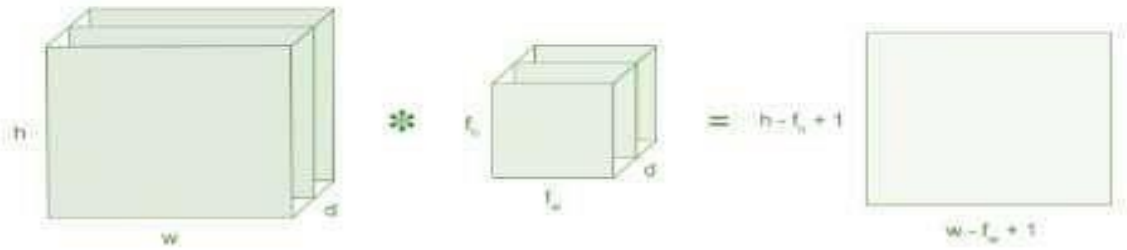
RGPVNOTES.IN



6. ***Convolution Layer***

Convolution is the first layer to extract features from an input image. Convolution preserves the relationship between pixels by learning image features using small squares of input data. It is a mathematical operation that takes two inputs such as image matrix and a filter or kernel.

- An image matrix (volume) of dimension **(h x w x d)**
- A filter **(f_h x f_w x d)**
- Outputs a volume dimension **(h - f_h + 1) x (w - f_w + 1) x 1**



Consider a 5 x 5 whose image pixel values are 0, 1 and filter matrix 3 x 3 as shown in below

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

5 x 5 – Image Matrix

*

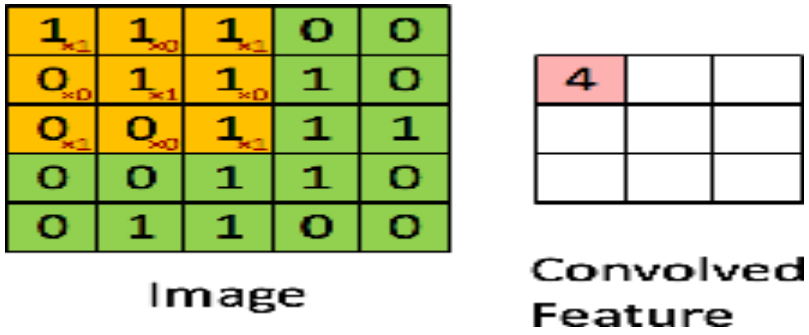
1	0	1
0	1	0
1	0	1

3 x 3 – Filter Matrix

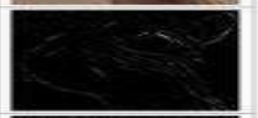
Then the convolution of 5 x 5 image matrix multiplies with 3 x 3 filter matrix which is called **"Feature Map"** as output shown in below

Get real-time updates from RGPV

RGPVNOTES.IN



Convolution of an image with different filters can perform operations such as edge detection, blur and sharpen by applying filters. The below example shows various convolution image after applying different types of filters (Kernels).

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	
Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

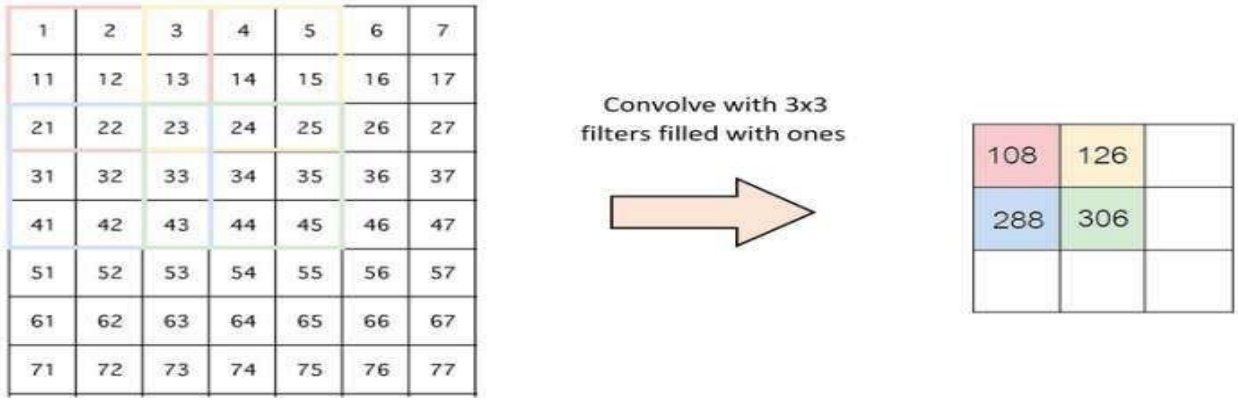
Stream Tech Notes

Strides

Stride is the number of pixels shifts over the input matrix. When the stride is 1 then we move the filters to 1 pixel at a time. When the stride is 2 then we move the filters to 2 pixels at a time and so on. The below figure shows convolution would work with a stride of 2.

Get real-time updates from RGPV

RGPVNOTES.IN



Padding

Sometimes filter does not fit perfectly fit the input image. We have two options:

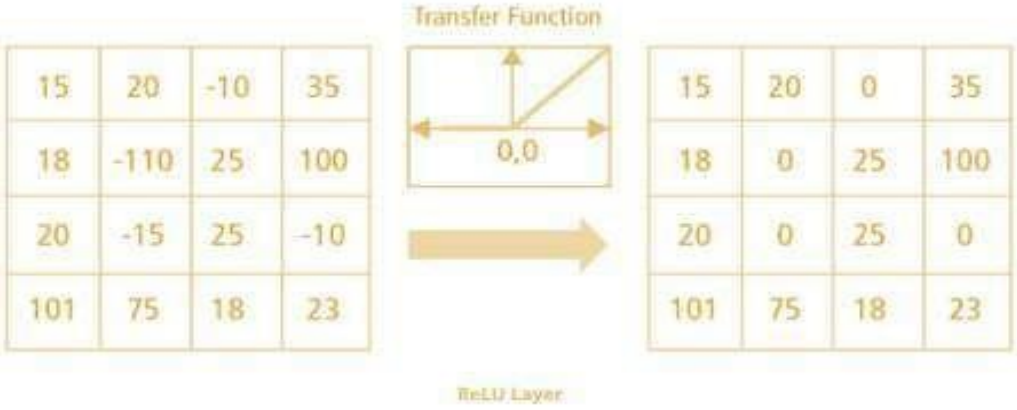
- Pad the picture with zeros (zero-padding) so that it fits
- Drop the part of the image where the filter did not fit. This is called valid padding which keeps only valid part of the image.

ch Notes

Non Linearity (ReLU)

ReLU stands for Rectified Linear Unit for a non-linear operation. The output is ***f(x) = max(0,x).***

Why ReLU is important : ReLU's purpose is to introduce non-linearity in our ConvNet. Since, the real world data would want our ConvNet to learn would be non-negative linear values.



There are other non linear functions such as tanh or sigmoid that can also be used instead of ReLU. Most of the data scientists use ReLU since performance wise ReLU is better than the other two.

7. Pooling Layer

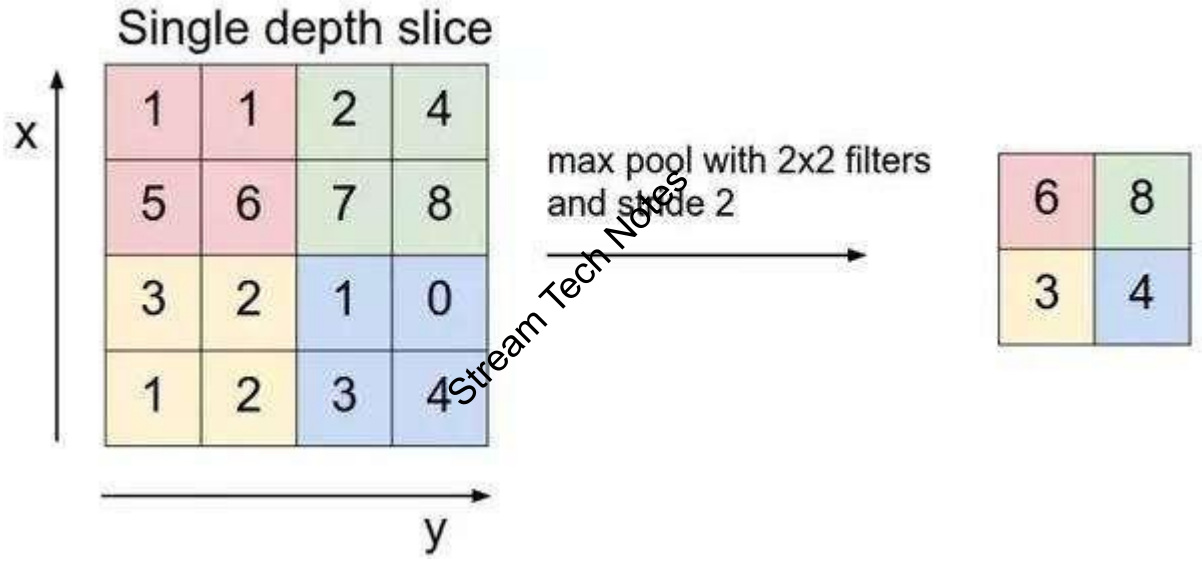
Get real-time updates from RGPV

RGPVNOTES.IN

Pooling layers section would reduce the number of parameters when the images are too large. Spatial pooling also called subsampling or downsampling which reduces the dimensionality of each map but retains important information. Spatial pooling can be of different types:

- Max Pooling
- Average Pooling
- Sum Pooling

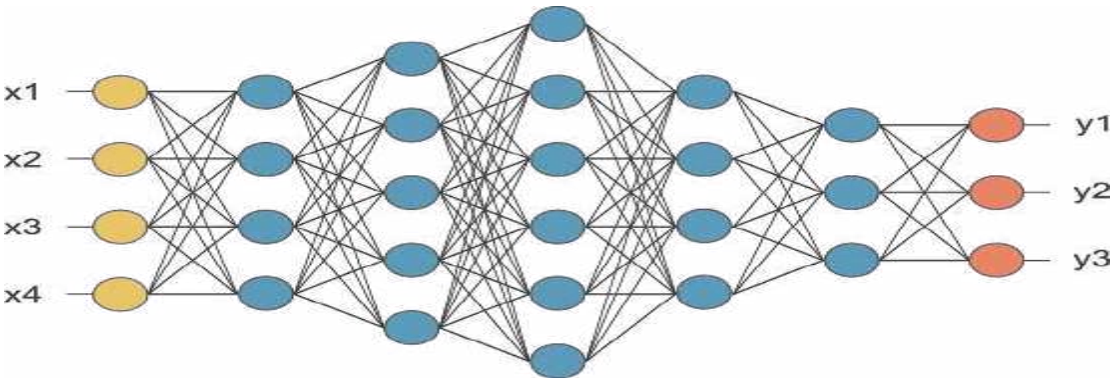
Max pooling takes the largest element from the rectified feature map. Taking the largest element could also take the average pooling. Sum of all elements in the feature map call as sum pooling.



Stream Tech Notes

Fully Connected Layer

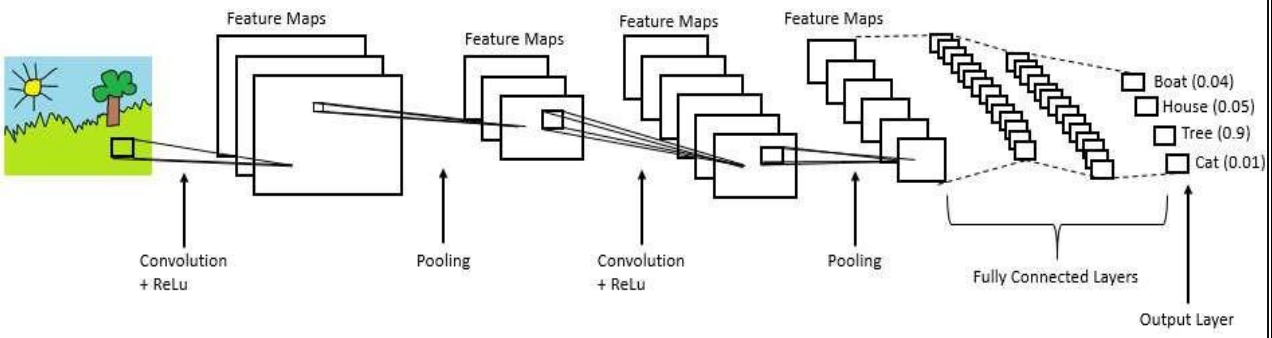
The layer we call as FC layer, we flattened our matrix into vector and feed it into a fully connected layer like a neural network.



Get real-time updates from RGPV

RGPVNOTES.IN

In the above diagram, the feature map matrix will be converted as vector (x1, x2, x3, …). With the fully connected layers, we combined these features together to create a model. Finally, we have an activation function such as softmax or sigmoid to classify the outputs as cat, dog, car, truck etc.,



Summary

Stream Tech Notes

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

In the next post, I would like to talk about some popular CNN architectures such as AlexNet, VGGNet, GoogLeNet, and ResNet.

8. loss layer : (Missing)

9. ***1x1 convolutions***

Convolutions layers are lighter than fully connected ones. But they still connect every input channels with every output channels for every position in the kernel windows. This is what

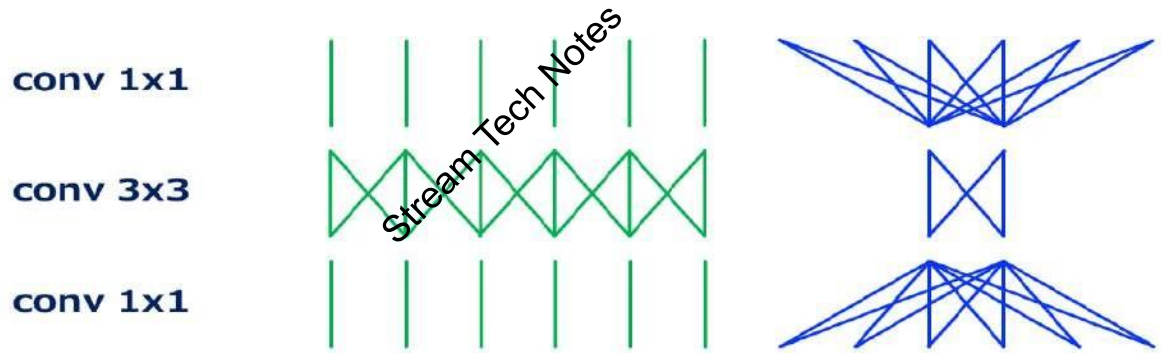
Get real-time updates from RGPV

RGPVNOTES.IN

gives the c_in * c_out multiplicative factor in the number of weights.

ILSVRC's Convnets use a lot of channels. 512 channels are used in VGG16's convolutional layers for example.
1x1 convolution is a solution to compensate for this.
It obviously doesn't bring anything at the spatial level: the kernel acts on one pixel at a time. But it acts as a fully connected layer pixel-wise.
We would usually have a 3x3 kernel size with 256 input and output channels. Instead of this, we first do a 1x1 convolutional layer bringing the number of channels down to something like 32. Then we perform the convolution with a 3x3 kernel size. We finally make another 1x1 convolutional layer to have 256 channels again.

The first solution needs $3^2 * 256^2 = 65{,}536$ weights. The second one needs $1^2 * 256 * 32 + 3^2 * 32^2 + 1^2 * 32 * 256 = 25{,}600$ weights. The convolution kernel is more than 2 times lighter.
A 1x1 convolution kernel acts as an embedding solution. It reduces the size of the input vector, the number of channels. It makes it more meaningful. The 1x1 convolutional layer is also called a Pointwise Convolution.



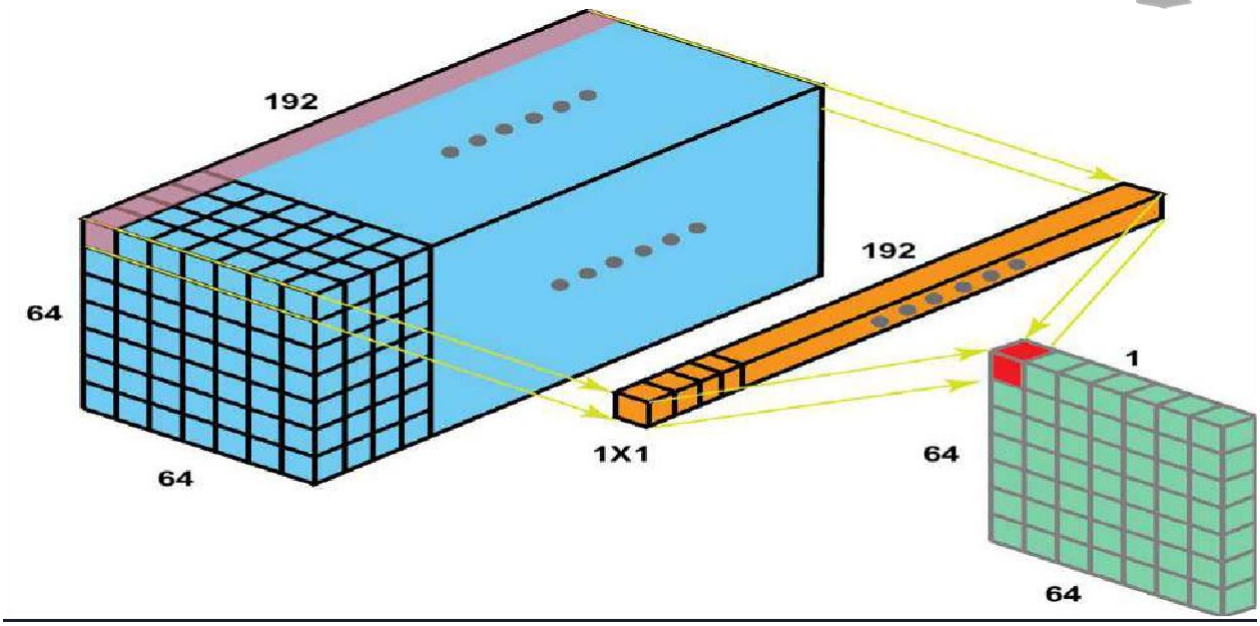
Spatial (green) and layer (blue) connections in a bottleneck.

In Details (for more understanding)

1 x 1 conv was used to reduce the number of channels while introducing non-linearity.
In 1X1 Convolution simply means the filter is of size 1X1 (Yes — that means a single number as opposed to matrix like, say 3X3 filter). This 1X1 filter will convolve over the ENTIRE input image pixel by pixel.
Staying with our example input of 64X64X3, if we choose a 1X1 filter (which would be 1X1X3), then the output will have the same Height and Weight as input but only one channel — 64X64X1
Now consider inputs with large number of channels — 192 for example. If we want to reduce the depth and but keep the Height X Width of the feature maps (Receptive field) the same, then we can choose 1X1 filters (remember Number of filters = Output Channels) to achieve this effect. This effect of cross channel down-sampling is called 'Dimensionality reduction'.

Get real-time updates from RGPV

RGPVNOTES.IN



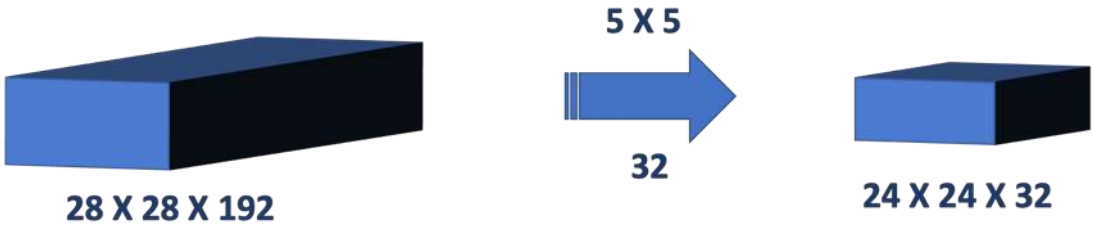
Now why would we want to something like that? For that we delve into usage of 1X1 Convolution

Usage 1: Dimensionality Reduction/Augmentation

~cr Notes

Winner of ILSVRC (ImageNet Large Scale Visual Recognition Competition) 2014, GoogleNet, used 1X1 convolution layer for dimension reduction "to compute reductions before the expensive 3×3 and 5×5 convolutions"

Let us look at an example to understand how reducing dimension will reduce computational load. Suppose we need to convolve 28 X 28 X 192 input feature maps with 5 X 5 X 32 filters. This will result in 120.422 Million operations

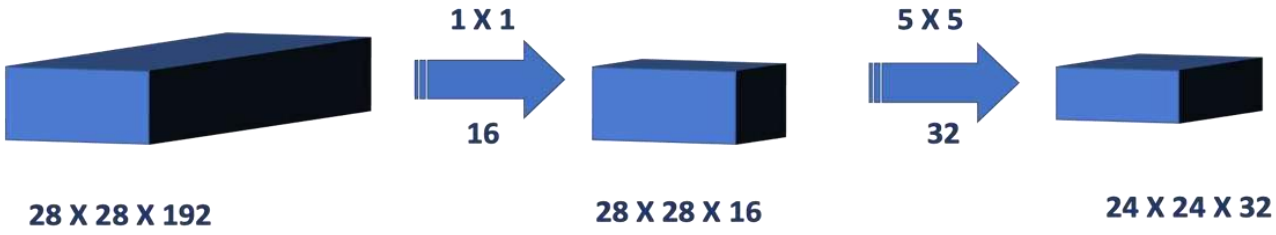


Number of Operations : (28X28X32) X (5X5X192) = 120.422 Million Ops

Let us do some math with the same input feature maps but with 1X1 Conv layer before the 5 X 5 conv layer

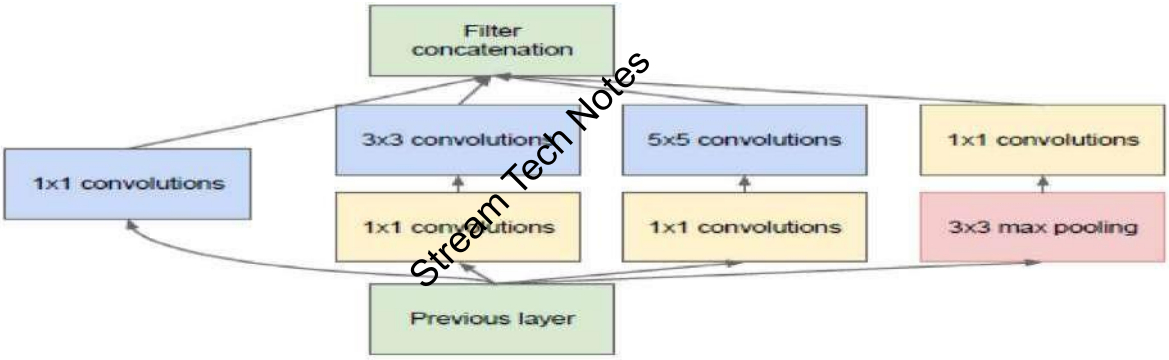
Get real-time updates from RGPV

RGPVNOTES.IN



Number of Operations for 1 X 1 Conv Step : (28X28X16) X (1X1X192) = 2.4 Million Ops
Number of Operations for 5 X 5 Conv Step : (28X28X32) X (5X5X16) = 10 Million Ops
Total Number of Operations = 12.4 Million Ops

By adding 1X1 Conv layer before the 5X5 Conv, while keeping the height and width of the feature map, we have reduced the **number of operations by a factor of 10**. **This will reduce the computational needs and in turn will end up being more efficient.**
GoogleNet paper describes the module as "**Inception Module**"



(b) Inception module with dimensionality reduction

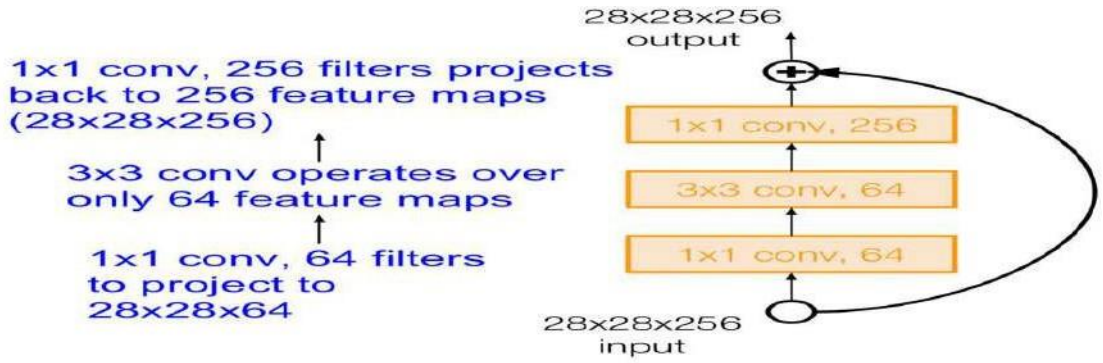
Stream Tech Notes

Usage 2: Building DEEPER Network ("Bottle-Neck" Layer)

2015 ILSVRC Classification winner, **ResNet,** had least error rate and swept aside the competition by using very deep network using 'Residual connections' and 'Bottle-neck Layer'.
In their paper, He et all explains (page 6) how a bottle neck layer designed **using a sequence of 3 convolutional layers with filters the size of 1X1, 3X3, followed by 1X1 respectively to reduce and restore dimension**. The down-sampling of the input happens in 1X1 layer thus funneling a smaller feature vectors (reduced number of parameters) for the 3X3 conv to work on.
Immediately after that 1X1 layer restores the dimensions to match input dimension so identity shortcuts can be directly used. For details on identity shortcuts and skip connection, please see some of the Reviews on ResNet (Or you can wait for my future work!)

Get real-time updates from RGPV

RGPVNOTES.IN



Usage 3: Smaller yet Accurate Model ("FIRE-MODULE" Layer)

While Deep CNN Models have great accuracy, they have staggering number of parameters to deal with which increases the training time and most importantly need enterprise level computing power. Iandola et all proposed a CNN Model called **Squeeze Net** that retains AlexNet level accuracy while 50X times smaller in terms of parameters.

StreamTechNotes

Smaller models have number of advantages especially on use-cases that require edge computing capabilities like autonomous driving. Iandola et all achieved this by stacking a bunch of "**Fire Modules**" which comprise of

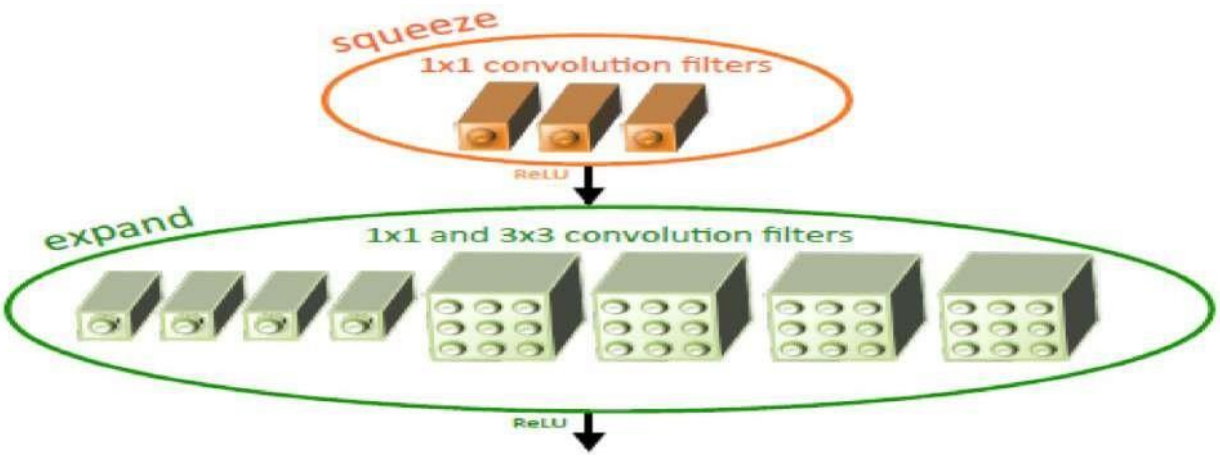
1. Squeeze Layer which has only 1X1 Conv filters
2. This feeds an Expansion layer which has mix of 1X1 and 3X3 filters
3. The number of filters in Squeeze Layer are set to be less than number of 1X1 filters + Number of 3X3 in Expand Layer

By now it is obvious what the 1X1 Conv filters in Squeeze Layer do — they reduce the number of parameters by 'down-sampling' the input channels before they are fed into the Expand layer.

The Expansion Layer has mix of 1X1 and 3X3 filters. The 1X1 filters, as you know, performs cross channel pooling — Combines channels, but cannot detect spatial structures (by virtue of working on individual pixels as opposed to a patch of input like larger filters). The 3X3 Convolution detects spatial structures. By combining these 2 different sized filters, the model becomes more expressive while operating on lesser parameters. Appropriate use of padding makes the output of 1X1 and 3X3 convolutions the same size so these can be stacked.

Get real-time updates from RGPV

RGPVNOTES.IN



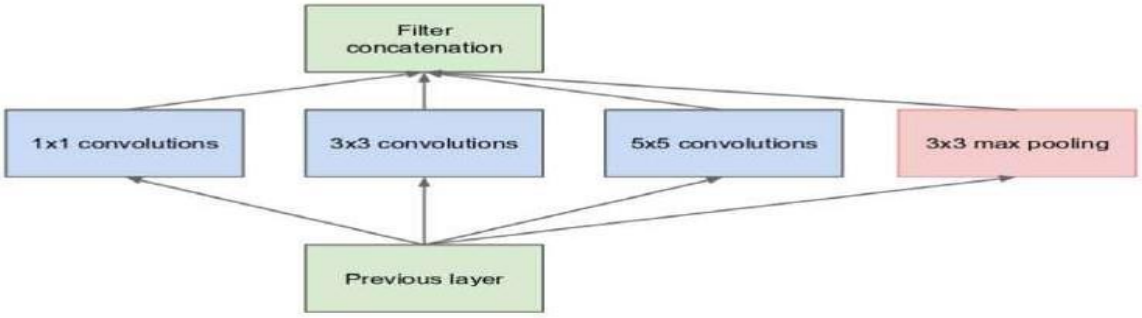
1X1 Convolution is effectively used for

1. Dimensionality Reduction/Augmentation
2. Reduce computational load by reducing parameter map
3. Add additional **non-linearity** to the network
4. Create deeper network through "Bottle-Neck" layer
5. Create smaller CNN network which retains higher degree of accuracy

Stream Tech Notes

10. What is an Inception Module?

Inception Modules are used in Convolutional Neural Networks to allow for more efficient computation and deeper Networks through a dimensionality reduction with stacked 1×1 convolutions. The modules were designed to solve the problem of computational expense, as well as overfitting, among other issues. The solution, in short, is to take multiple kernel filter sizes within the CNN, and rather than stacking them sequentially, ordering them to operate on the same level.



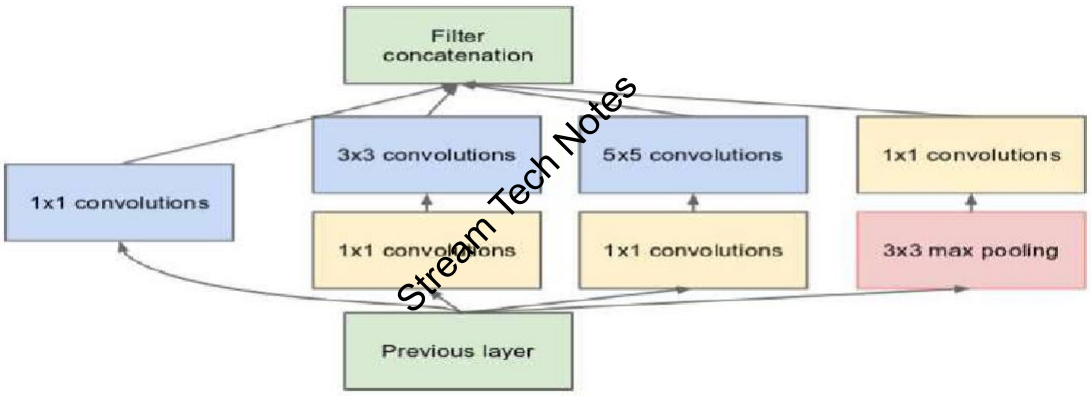
(a) Inception module, naïve version

Get real-time updates from RGPV

RGPVNOTES.IN

How does an Inception Module work?

Inception Modules are incorporated into convolutional neural networks (CNNs) as a way of reducing computational expense. As a neural net deals with a vast array of images, with wide variation in the featured image content, also known as the salient parts, they need to be designed appropriately. The most simplified version of an inception module works by performing a convolution on an input with not one, but three different sizes of filters (1x1, 3x3, 5x5). Also, max pooling is performed. Then, the resulting outputs are concatenated and sent to the next layer. By structuring the CNN to perform its convolutions on the same level, the network gets progressively wider, not deeper.

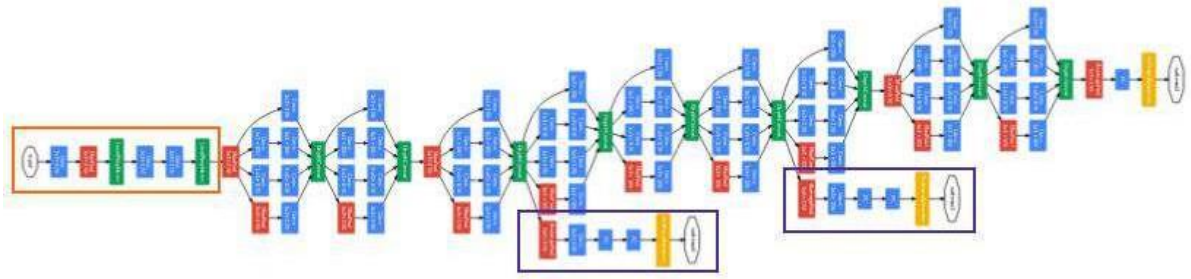


(b) Inception module with dimension reductions

To make the process even less computationally expensive, the neural network can be designed to add an extra 1x1 convolution before the 3x3 ad 5x5 layers. By doing so, the number of input channels is limited and 1x1 convolutions are far cheaper than 5x5 convolutions. It is important to note, however, that the 1x1 convolution is added after the max-pooling layer, rather than before.

Get real-time updates from RGPV

RGPVNOTES.IN



The design of this initial Inception Module is known commonly as GoogLeNet, or Inception v1. Additional variations to the inception module have been designed, reducing issues such as the vanishing gradient problem.

11. Input Channels

Stream Tech Notes

Imagine a network as a sequence of "layers", where each layer is of the form xn+1=f(xn)xn+1=f(xn), where f(x)f(x) is a linear transformation followed by a non-linearity such as sigmoid, tanh or relu. The layers operate on 3-D chunks of data, where the first two dimensions are (generally) the height and width of an image patch, and the third dimension is a number of such patches stacked over one another, which is also called the number of **channels** in the image volume. Thus, xx can be viewed as a H×W×CH×W×C vector, where H,WH,W are the dimensions of the image and CC is the number of channels of the given image volume.

Multiple Input and Output channels

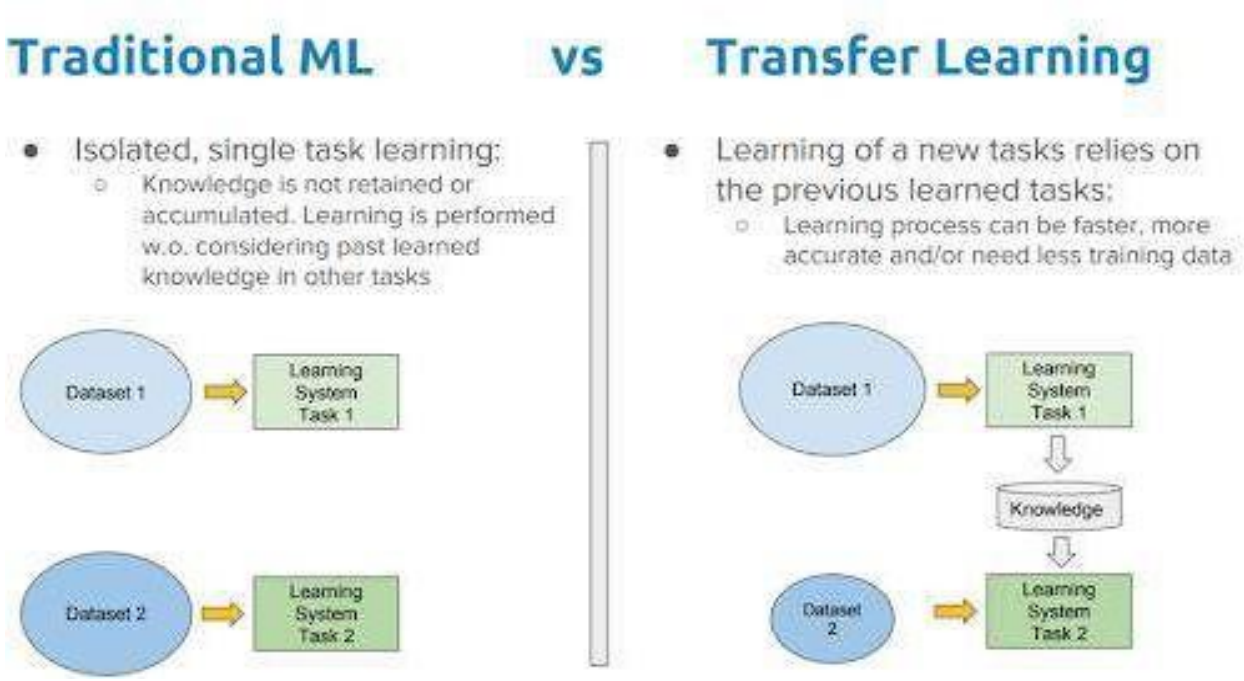
While we have described the multiple channels that comprise each image (e.g., color images have the standard RGB channels to indicate the amount of red, green and blue), until now, we simplified all of our numerical examples by working with just a single input and a single output channel. This has allowed us to think of our inputs, convolutional kernels, and outputs each as two-dimensional arrays. When we add channels into the mix, our inputs and hidden representations both become three-dimensional arrays. For example, each RGB input image has shape 3×h×w3×h×w. We refer to this axis, with a size of 3, as the channel dimension.

12. Transfer learning

Transfer learning is the idea of overcoming the isolated learning paradigms and utilizing the knowledge acquired for one task to solve related ones, as applied to machine learning, and in particular, to the domain of deep learning.

Get real-time updates from RGPV

RGPVNOTES.IN



Transfer Learning for Deep Learning Networks

Stream Tech Notes

Why transfer learning ?

Many deep neural networks trained on natural images exhibit a curious phenomenon in common: on the first layer they learn features similar to Gabor filters and color blobs. Such first-layer features appear not to specific to a particular dataset or task but are general in that they are applicable to many datasets and tasks. As finding these standard features on the first layer seems to occur regardless of the exact cost function and natural image dataset, we call these first-layer features general. For example, in a network with an N-dimensional softmax output layer that has been successfully trained towards a supervised classification objective, each output unit will be specific to a particular class. We thus call the last-layer features specific.

In transfer learning we first train a base network on a base dataset and task, and then we repurpose the learned features, or transfer them, to a second target network to be trained on a target dataset and task. This process will tend to work if the features are general, that is, suitable to both base and target tasks, instead of being specific to the base task.

In practice, very few people train an entire Convolutional Network from scratch because it is relatively rare to have a dataset of sufficient size. Instead, it is common to pre-train a ConvNet on a very large dataset (e.g. ImageNet, which contains 1.2 million images with 1000 categories), and then use the ConvNet either as an initialization or a fixed feature extractor for the task of interest. There is a myriad of strategies to follow for the transfer learning process in the deep

Therefore, a widely used strategy in transfer learning is to:

Get real-time updates from RGPV

RGPVNOTES.IN

- Load the weights matrices of a pre-trained model except for the weights of the very last layers near the O/P,
- Hold those weights fixed, i.e. untrainable
- Attach new layers suitable for the task at hand, and train the model with new data

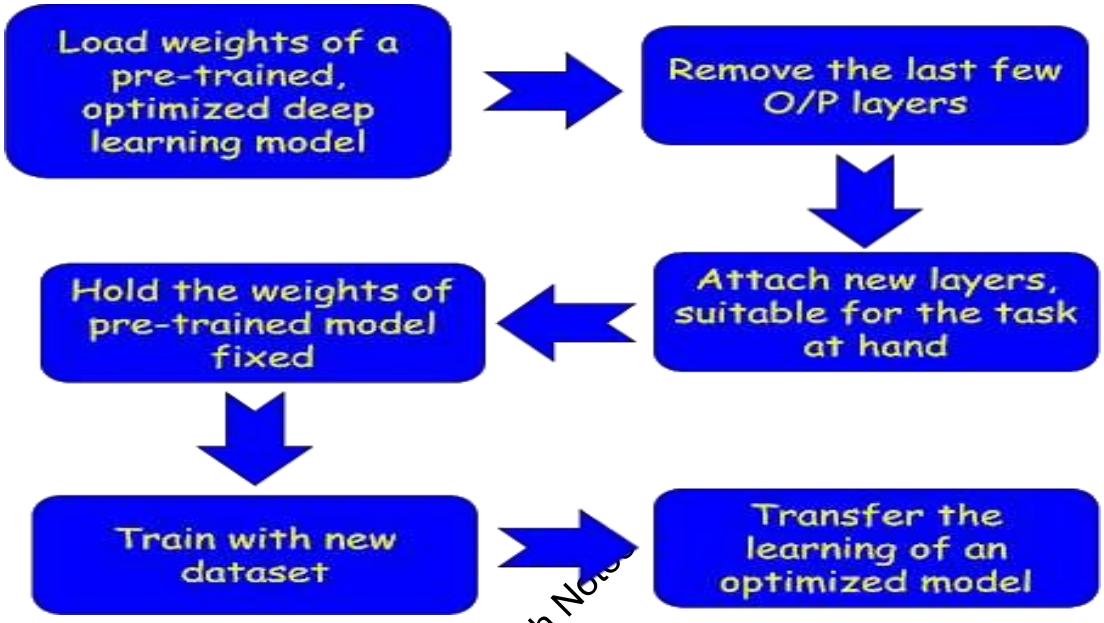


Fig: The transfer learning strategy for deep learning networks, as we will explore here.

StreamTech Notes

This way, we don't have to train the whole model, we get to repurpose the model for our specific machine learning task, yet can leverage the learned structures and patterns of the data, contained in the fixed weights, which are loaded from the pre-trained, optimized model.

13. One shot learning

Deep Convolutional Neural Networks have become the state of the art methods for image classification tasks. However, one of the biggest limitations is they require a lots of labelled data. In many applications, collecting this much data is sometimes not feasible. One Shot Learning aims to solve this problem.

Classification vs One Shot Learning

In case of **standard classification**, the input image is fed into a series of layers, and finally at the output we generate a probability distribution over all the classes (typically using a Softmax). For example, if we are trying to classify an image as cat or dog or horse or elephant, then for every input image, we generate 4 probabilities, indicating the probability of the image belonging to each of the 4 classes. Two important points must be noticed here. **First**, during the training process, we require a **large** number of images for each of the class (cats, dogs, horses and elephants). **Second**, if the network is trained only on the above 4 classes of images, then we cannot expect to test it on any other class, example "zebra". If we want our model to classify the images of zebra as well,

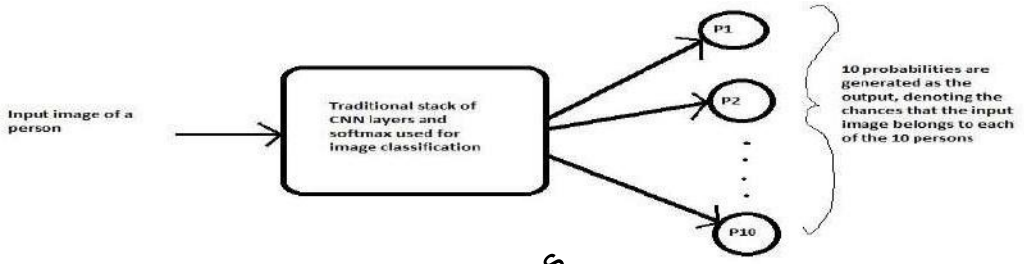
Get real-time updates from RGPV

RGPVNOTES.IN

then we need to first get a lot of zebra images and then we must **re-train** the model again. There are applications wherein we neither have enough data for each class and the total number classes is huge as well as dynamically changing. Thus, the cost of data collection and periodical re-training is too high.

On the other hand, in a **one shot classification**, we require only one training example for each class. Yes you got that right, just one. Hence the name **One Shot**. Let's try to understand with a real world practical example.

Assume that we want to build face recognition system for a small organization with only 10 employees (small numbers keep things simple). Using a traditional classification approach, we might come up with a system that looks as below:



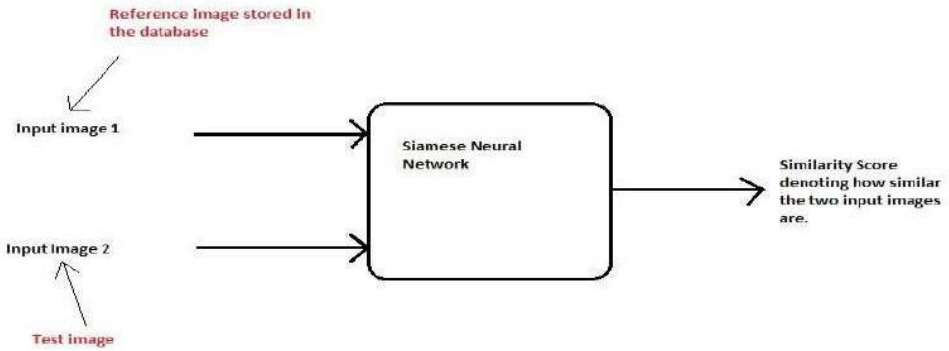
Stream Tech Notes

Problems:

a) To train such a system, we first require a lot of **different** images of each of the 10 persons in the organization which might not be feasible. (Imagine if you are doing this for an organization with thousands of employees).

b) What if a new person joins or leaves the organization? You need to take the pain of collecting data again and re-train the entire model again. This is practically not possible specially for large organizations where recruitment and attrition is happening almost every week.

Now let's understand how do we approach this problem using one shot classification which helps to solve both of the above issues:



Get real-time updates from RGPV

RGPVNOTES.IN

Instead of directly classifying an input(test) image to one of the 10 people in the organization, this network instead takes an extra reference image of the person as input and will produce a similarity score denoting the chances that the two input images belong to the same person. Typically the similarity score is squished between 0 and 1 using a sigmoid function; wherein 0 denotes no similarity and 1 denotes full similarity. Any number between 0 and 1 is interpreted accordingly.
Notice that this network is not learning to classify an image directly to any of the output classes. Rather, it is learning a **similarity function**, which takes two images as input and expresses how similar they are.
How does this solve the two problems we discussed above?
a) In a short while we will see that to train this network, you do not require too many instances of a class and only few are enough to build a good model.
b) But the biggest advantage is that , let's say in case of face recognition, we have a new employee who has joined the organization. Now in order for the network to detect his face, we only require a **single** image of his face which will be stored in the database. Using this as the reference image, the network will calculate the similarity for any new instance presented to it. Thus we say that network predicts the score in **one shot**.

14. What is Dimensionality Reduction?

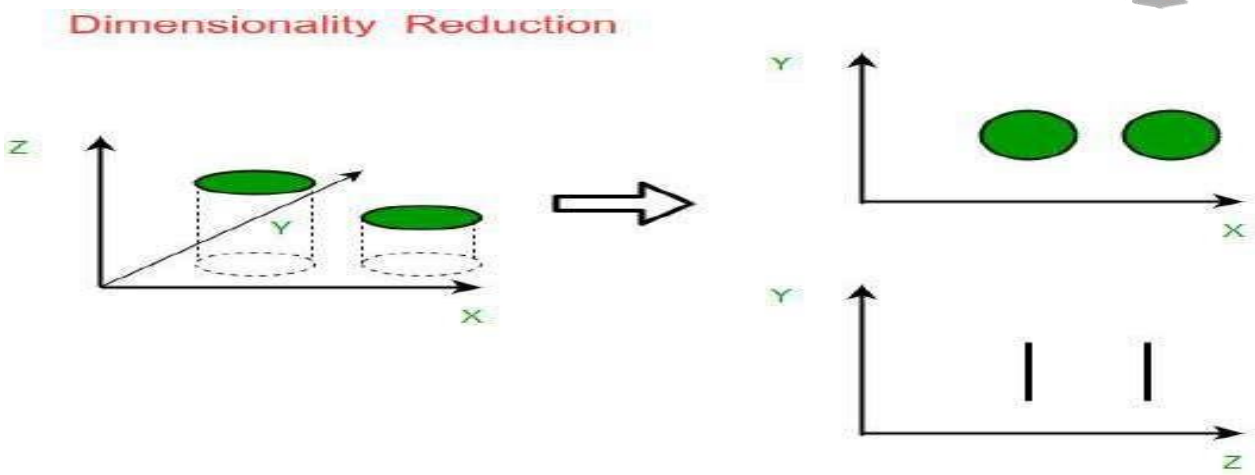
Stream Tech Notes

What is Dimensionality Reduction?
In machine learning classification problems, there are often too many factors on the basis of which the final classification is done. These factors are basically variables called features. The higher the number of features, the harder it gets to visualize the training set and then work on it. Sometimes, most of these features are correlated, and hence redundant. This is where dimensionality reduction algorithms come into play. Dimensionality reduction is the process of reducing the number of random variables under consideration, by obtaining a set of principal variables. It can be divided into feature selection and feature extraction.

Why is Dimensionality Reduction important in Machine Learning and Predictive Modeling?
An intuitive example of dimensionality reduction can be discussed through a simple e-mail classification problem, where we need to classify whether the e-mail is spam or not. This can involve a large number of features, such as whether or not the e-mail has a generic title, the content of the e-mail, whether the e-mail uses a template, etc. However, some of these features may overlap. In another condition, a classification problem that relies on both humidity and rainfall can be collapsed into just one underlying feature, since both of the aforementioned are correlated to a high degree. Hence, we can reduce the number of features in such problems. A 3-D classification problem can be hard to visualize, whereas a 2-D one can be mapped to a simple 2 dimensional space, and a 1-D problem to a simple line. The below figure illustrates this concept, where a 3-D feature space is split into two 1-D feature spaces, and later, if found to be correlated, the number of features can be reduced even further.

Get real-time updates from RGPV

RGPVNOTES.IN



There are two components of dimensionality reduction:

- Feature selection: In this, we try to find a subset of the original set of variables, or features, to get a smaller subset which can be used to model the problem. It usually involves three ways:
    1. Filter
    2. Wrapper
    3. Embedded
- Feature extraction: This reduces the data in a high dimensional space to a lower dimension space, i.e. a space with lesser no. of dimensions.

StreamTech Notes

Methods of Dimensionality Reduction

The various methods used for dimensionality reduction include:

- Principal Component Analysis (PCA)
- Linear Discriminant Analysis (LDA)
- Generalized Discriminant Analysis (GDA)

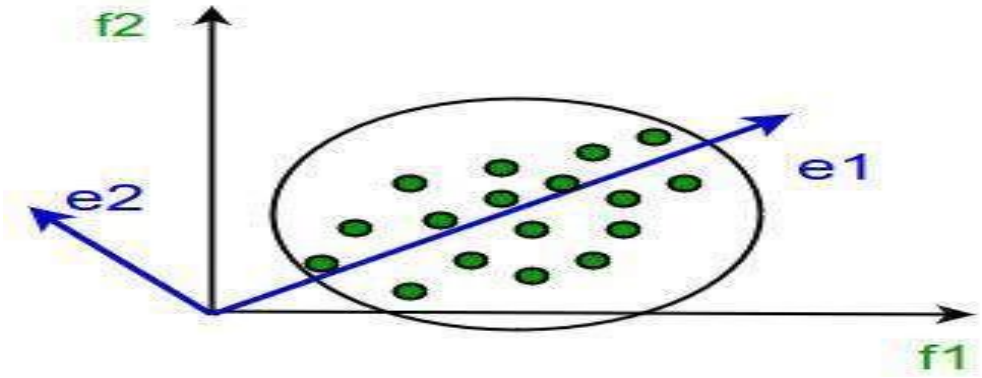
Dimensionality reduction may be both linear or non-linear, depending upon the method used. The prime linear method, called Principal Component Analysis, or PCA, is discussed below.

- **Principal Component Analysis**

This method was introduced by Karl Pearson. It works on a condition that while the data in a higher dimensional space is mapped to data in a lower dimension space, the variance of the data in the lower dimensional space should be maximum.

Get real-time updates from RGPV

RGPVNOTES.IN



It involves the following steps:

- Construct the covariance matrix of the data.
- Compute the eigenvectors of this matrix.
- Eigenvectors corresponding to the largest eigenvalues are used to reconstruct a large fraction of variance of the original data.

Hence, we are left with a lesser number of eigenvectors, and there might have been some data loss in the process. But, the most important variances should be retained by the remaining eigenvectors.

Stream Tech Notes

Advantages of Dimensionality Reduction

- It helps in data compression, and hence reduced storage space.
- It reduces computation time.
- It also helps remove redundant features, if any.

Disadvantages of Dimensionality Reduction

- It may lead to some amount of data loss.
- PCA tends to find linear correlations between variables, which is sometimes undesirable.
- PCA fails in cases where mean and covariance are not enough to define datasets.
- We may not know how many principal components to keep- in practice, some thumb rules are applied.

- ***Other methods to perform Dimension Reduction?(Optional)***

There are many methods to perform Dimension reduction. I have listed the most common methods below:

1. Missing Values: While exploring data, if we encounter missing values, what we do? Our first step should be to identify the reason then impute missing values/ drop variables using appropriate methods. But, what if we have too many missing values? Should we impute missing values or drop the variables?

I would prefer the latter, because it would not have lot more details about data set. Also, it would not help in improving the power of model. Next question, is there any threshold of missing values for dropping a variable? It varies from case to case. If the information contained in the variable is not that much, you can drop the variable if it has more than ~40-50% missing values.

Get real-time updates from RGPV

RGPVNOTES.IN

2. Low Variance: Let's think of a scenario where we have a constant variable (all observations have same value, 5) in our data set. Do you think, it can improve the power of model? Ofcourse NOT, because it has zero variance. In case of high number of dimensions, we should drop variables having low variance compared to others because these variables will not explain the variation in target variables.

3. Decision Trees: It is one of my favorite techniques. It can be used as a ultimate solution to tackle multiple challenges like missing values, outliers and identifying significant variables. It worked well in our Data Hackathon also. Several data scientists used decision tree and it worked well for them.

4. Random Forest: Similar to decision tree is Random Forest. I would also recommend using the in-built feature importance provided by random forests to select a smaller subset of input features. Just be careful that random forests have a tendency to bias towards variables that have more no. of distinct values i.e. favor numeric variables over binary/categorical values.

5. High Correlation: Dimensions exhibiting higher correlation can lower down the performance of model. Moreover, it is not good to have multiple variables of similar information or variation also known as **"Multicollinearity".** You can use *Pearson* (continuous variables) or *Polychoric* (discrete variables) correlation matrix to identify the variables with high correlation and select one of them using VIF (Variance Inflation Factor). Variables having higher value (VIF > 5) can be dropped.

Stream Tech Notes

15.Convolutional neural network?

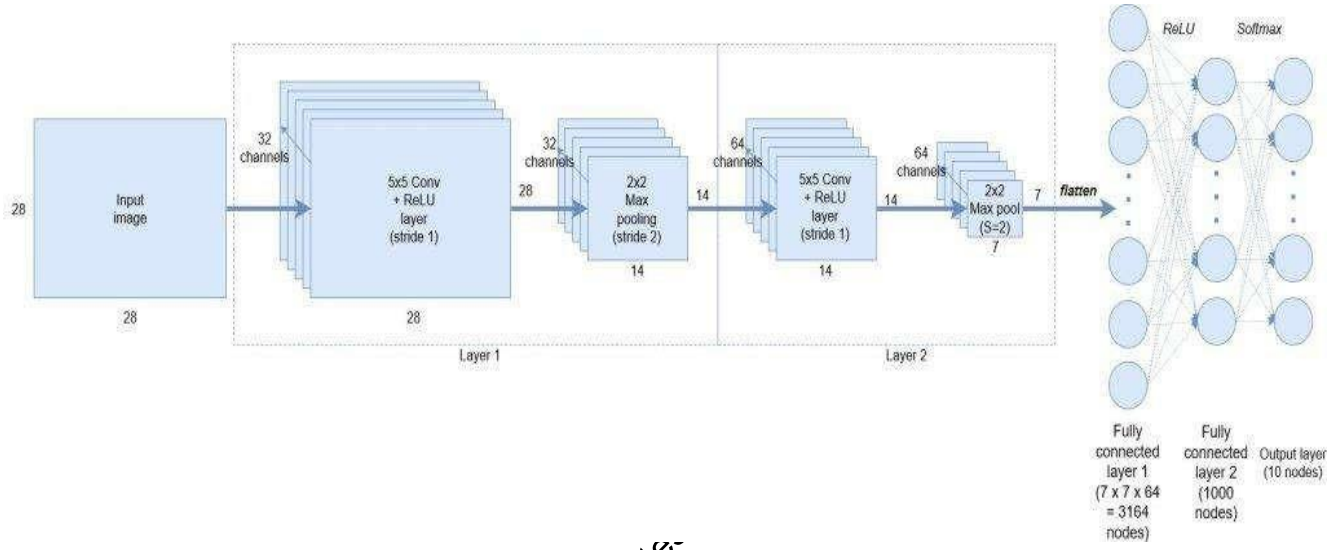
Multi-layer neural networks can perform pretty well in predicting things like digits in the MNIST dataset. This is especially true if we apply some improvements. So why do we need any other architecture? Well, first off – the MNIST dataset is quite simple. The images are small (only 28 x 28 pixels), are single layered (i.e. greyscale, rather than a coloured 3 layer RGB image) and include pretty simple shapes (digits only, no other objects). Once we start trying to classify things in more complicated colour images, such as buses, cars, trains etc. , we run into problems with our accuracy. What do we do?Well, first, we can try to increase the number of layers in our neural network to make it *deeper*. That will increase the complexity of the network and allow us to model more complicated functions. However, it will come at a cost – the number of parameters (i.e. weights and biases) will rapidly increase. This makes the model more prone to overfitting and will prolong training times. In fact, learning such difficult problems can become intractable for normal neural networks. This leads us to a solution – convolutional neural networks.

- ***A TensorFlow based convolutional neural network***

Get real-time updates from RGPV

RGPVNOTES.IN

TensorFlow makes it easy to create convolutional neural networks once you understand some of the nuances of the framework's handling of them. we are going to create a convolutional neural network with the structure detailed in the image below. The network we are going to build will perform MNIST digit classification,

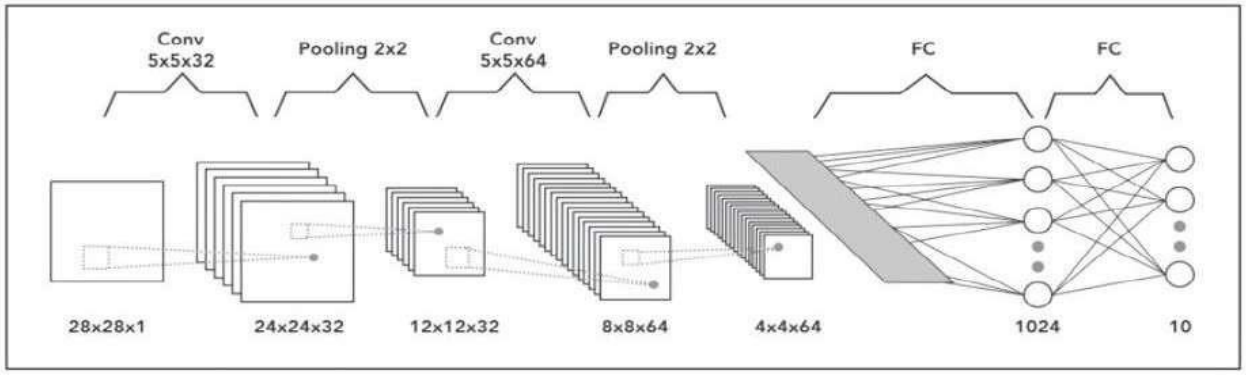


As can be observed, we start with the MNIST 28×28 greyscale images of digits. We then create 32, 5×5 convolutional filters / channels plus ReLU (rectified linear unit) node activations. After this, we still have a height and width of 28 nodes. We then perform down-sampling by applying a 2×2 max pooling operation with a stride of 2. Layer 2 consists of the same structure, but now with 64 filters / channels and another stride-2 max pooling down-sample. We then flatten the output to get a fully connected layer with 3164 nodes, followed by another hidden layer of 1000 nodes. These layers will use ReLU node activations. Finally, we use a softmax classification layer to output the 10 digit probabilities.

StreamTech Note-

- **Keras**

3- Convolutional Neural Network Architecture



A Simple Example of CNN Architecture

Get real-time updates from RGPV

RGPVNOTES.IN

First, we will define the Convolutional neural networks architecture as follows:

1- The first hidden layer is a convolutional layer called a Convolution2D. We will use 32 filters with size 5×5 each.

2- Then a Max pooling layer with a pool size of 2×2.

3- Another convolutional layer with 64 filters with size 5×5 each.

4- Then a Max pooling layer with a pool size of 2×2.

5- Then next is a Flatten layer that converts the 2D matrix data to a 1D vector before building the fully connected layers.

6- After that we will use a fully connected layer with 1024 neurons and relu activation function.

7- Then we will use a regularization layer called Dropout. It is configured to randomly exclude 20% of neurons in the layer in order to reduce overfitting.

8- Finally, the output layer which has 10 neurons for the 10 classes and a softmax activation function to output probability-like predictions for each class.

Stream Tech Notes

After deciding the above, we can set up a neural network model with a few lines of code as follows:

Step 1 – Create a model:
Keras first creates a new instance of a model object and then add layers to it one after the another. It is called a sequential model API. We can add layers to the neural network just by calling model.add and passing in the type of layer we want to add. Finally, we will compile the model with two important information, loss function, and cost optimization algorithm.

```
# Importing the required Keras modules containing model and layers

from keras.models import Sequential
from keras.layers import Dense, Conv2D, Dropout, Flatten, MaxPooling2D

# Creating a Sequential Model and adding the layers

model = Sequential()
model.add(Conv2D(32, kernel_size=(5,5), input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, kernel_size=(5,5), input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten()) # Flattening the 2D arrays for fully connected layers
model.add(Dense(1024, activation=tf.nn.relu))
```


Get real-time updates from RGPV

RGPVNOTES.IN

```
model.add(Dropout(0.2))
model.add(Dense(10,activation=tf.nn.softmax))

#Compile the model

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Once we execute the above code, Keras will build a TensorFlow model behind the scenes.

Step 2 – Train the model:

We can train the model by calling model.fit and pass in the training data and the expected output. Keras will run the training process and print out the progress to the console. When training completes, it will report the final accuracy that was achieved with the training data.

```
model.fit(x=x_train,y=y_train, epochs=10)
```

Step 3 – Test the model:

We can test the model by calling model.evaluate and passing in the testing data set and the expected output.

```
test_error_rate = model.evaluate(x_test, y_test, verbose=0)
print("The mean squared error (MSE) for the test data set is: {}".format(test_error_rate))
```

Stream Notes

Step 4 – Save and Load the model:

Once we reach the optimum results we can save the model using model.save and pass in the file name. This file will contain everything we need to use our model in another program.

```
model.save("trained_model.h5")
```

Your model will be saved in the Hierarchical Data Format (HDF) with .h5 extension. It contains multidimensional arrays of scientific data.

We can load our previously trained model by calling the load model function and passing in a file name. Then we call the predict function and pass in the new data for predictions.

```
model = keras.models.load_model("trained_model.h5")
predictions = model.predict(new_data)
```

Summary

- We learned how to load the MNIST dataset and normalize it.
- We learned the implementation of CNN using Keras.
- We saw how to save the trained model and load it later for prediction.
- The accuracy is more than 98% which is way more what we achieved with the regular neural network.

Get real-time updates from RGPV

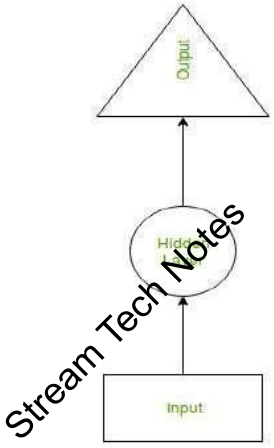
Downloaded from www.rgpvnotes.in

RGPVNOTES.IN

Department of Computer Science and Engineering
Subject Notes
CS 601- Machine Learning
Unit 4

Recurrent neural network

Recurrent Neural Network (RNN) are a type of Neural Network where the output from previous step are fed as input to the current step. In traditional neural networks, all the inputs and outputs are independent of each other, but in cases like when it is required to predict the next word of a sentence, the previous words are required and hence there is a need to remember the previous words. Thus, RNN came into existence, which solved this issue with the help of a Hidden Layer. The main and most important feature of RNN is Hidden state, which remembers some information about a sequence.



Stream Tech Notes

Figure : 4.1

RNN have a “memory” which remembers all information about what has been calculated. It uses the same parameters for each input as it performs the same task on all the inputs or hidden layers to produce the output. This reduces the complexity of parameters, unlike other neural networks.

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.

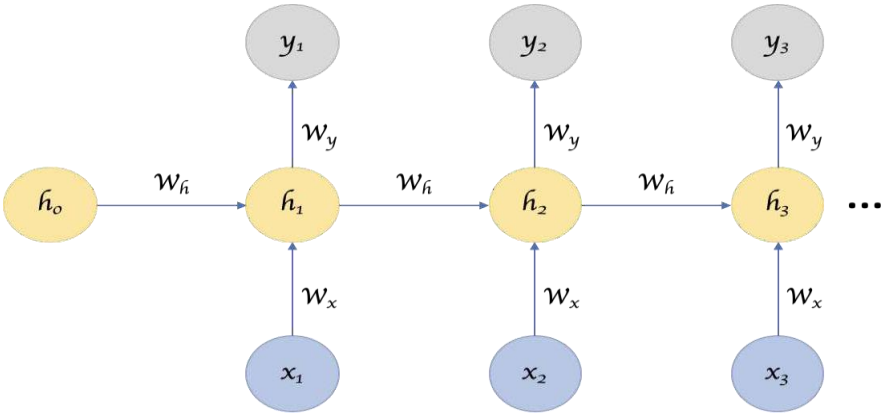


Figure :4.2 A Recurrent Neural Network, with a hidden state that is meant to carry pertinent information from one input item in the series to others.

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

RGPVNOTES.IN

The formula for the current state can be written as –

$$h_t = f(h_{t-1}, x_t)$$

Here, Ht is the new state, ht-1 is the previous state while xt is the current input. We now have a state of the previous input instead of the input itself, because the input neuron would have applied the transformations on our previous input. So each successive input is called as a time step.

Taking the simplest form of a recurrent neural network, let's say that the activation function is tanh, the weight at the recurrent neuron is Whh and the weight at the input neuron is Wxh, we can write the equation for the state at time t as –

$$h_t = \tanh (W_{hh}h_{t-1} + W_{xh}x_t)$$

The Recurrent neuron in this case is just taking the immediate previous state into consideration. For longer sequences the equation can involve multiple such states. Once the final state is calculated we can go on to produce the output

Now, once the current state is calculated we can calculate the output state as-

$$y_t = W_{hy}h_t$$

Stream Tech Notes

Training through RNN

1. A single time step of the input is supplied to the network i.e. xt is supplied to the network
2. We then calculate its current state using a combination of the current input and the previous state i.e. we calculate ht
3. The current ht becomes ht-1 for the next time step
4. We can go as many time steps as the problem demands and combine the information from all the previous states
5. Once all the time steps are completed the final current state is used to calculate the output yt
6. The output is then compared to the actual output and the error is generated
7. The error is then backpropagated to the network to update the weights (we shall go into the details of backpropagation in further sections) and the network is trained

Advantages of Recurrent Neural Network

1. An RNN remembers each and every information through time. It is useful in time series prediction only because of the feature to remember previous inputs as well. This is called Long Short-Term Memory.
2. Recurrent neural network is even used with convolutional layers to extend the effective pixel neighborhood.

Disadvantages of Recurrent Neural Network

1. Gradient vanishing and exploding problems.
2. Training an RNN is a very difficult task.
3. It cannot process very long sequences if using tanh or ReLu as an activation function.

Long short-term memory

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

RGPVNOTES.IN

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.

new weight = weight - learning rate*gradient

2.0999 = 2.1 - 0.001

Not much of a difference — update value

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.

Long Short Term Memory is a kind of recurrent neural network is the solution of the above problem. In RNN output from the last step is fed as input in the current step. LSTM was desgined by Hochreiter & Schmidhuber. It tackled the problem of long-term dependencies of RNN in which the RNN cannot predict the word stored in the long-term memory but can give more accurate predictions from the recent information. As the gap length increases RNN does not give efficent performance. LSTM can by default retain the information for long period of time. It is used for processing, predicting and classifying on the basis of time series data.

Stream Tech Notes

Structure Of LSTM:

LSTM has a chain structure that contains four neural networks and different memory blocks called **cells**.

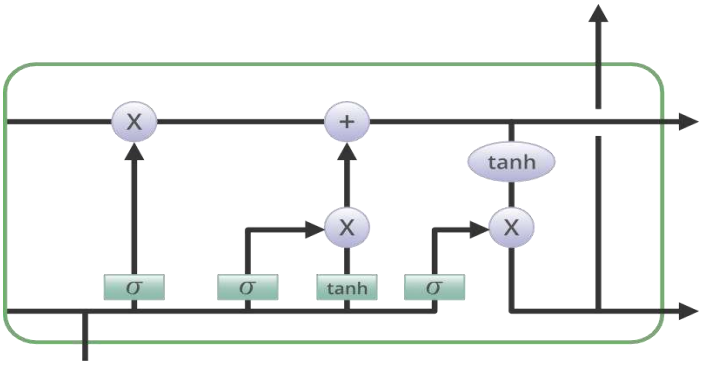


Figure: 4.3

Information is retained by the cells and the memory manipulations are done by the **gates.**

There are three gates –

1. **Forget Gate:** The information that no longer useful in the cell state is removed with the forget gate. Two inputs *x_t* (input at the particular time) and *h_t-1* (previous cell output) are fed to the gate and multiplied with weight matrices followed by the addition of bias. The resultant is passed through an activation function which gives a binary output. If for a particular cell state the output is 0, the piece of information is forgotten and for the output 1, the information is retained for the future use.

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

RGPVNOTES.IN

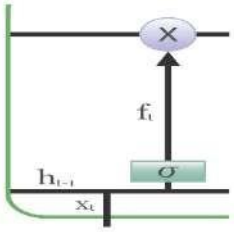
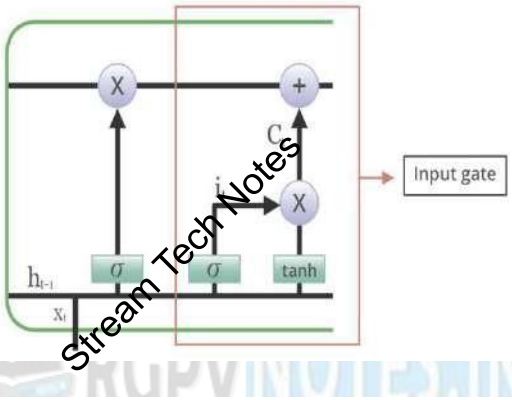


Figure: 4.4

2. **Input gate:** Addition of useful information to the cell state is done by input gate. First, the information is regulated using the sigmoid function and filter the values to be remembered similar to the forget gate using inputs *h_t-1* and *x_t*. Then, a vector is created using *tanh* function that gives output from -1 to +1, which contains all the possible values from h_t-1 and *x_t*. Atlast, the values of the vector and the regulated values are multiplied to obtain the useful information.



Stream Tech Notes

Figure: 4.5

3. **Output gate:** The task of extracting useful information from the current cell state to be presented as an output is done by output gate. First, a vector is generated by applying tanh function on the cell. Then, the information is regulated using the sigmoid function and filter the values to be remembered using inputs *h_t-1* and *x_t*. At last, the values of the vector and the regulated values are multiplied to be sent as an output and input to the next cell.

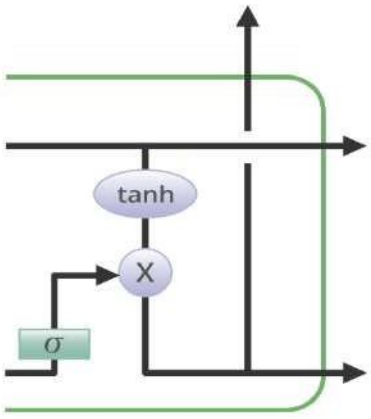


Figure: 4.6

Some of the famous applications of LSTM includes:

1. Language Modelling
2. Machine Translation

Get real-time updates from RGPV

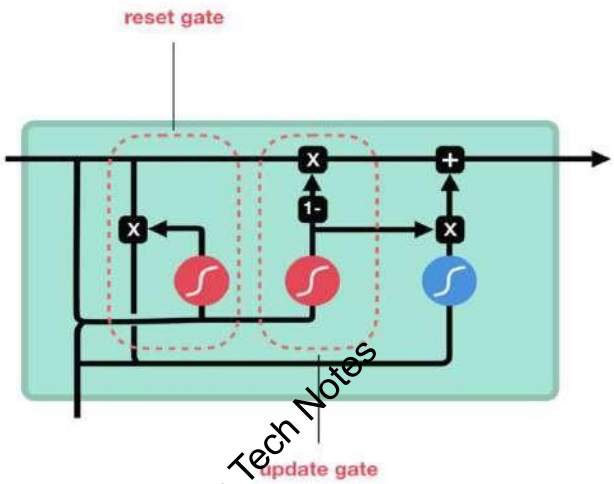
Downloaded from www.rgpvnotes.in

RGPVNOTES.IN

3. Image Captioning
4. Handwriting generation
5. Question Answering Chatbots

Gated recurrent unit

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.



Stream Tech Notes

Figure: 4.7

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.

Translation

One of the earliest goals for computers was the automatic translation of text from one language to another.

Automatic or machine translation is perhaps one of the most challenging artificial intelligence tasks given the fluidity of human language. Classically, rule-based systems were used for this task, which were replaced in the 1990s with statistical methods. More recently, deep neural network models achieve state-of-the-art results in a field that is aptly named neural machine translation.

Machine translation is the task of automatically converting source text in one language to text in another language.

In a machine translation task, the input already consists of a sequence of symbols in some language, and the computer program must convert this into a sequence of symbols in another language. Given a sequence of text in a source language, there is no one single best translation of that text to another language. This is because of the natural ambiguity and flexibility of human language. The fact is that accurate translation requires background knowledge in order to resolve ambiguity and establish the content of the sentence.

Classical machine translation methods often involve rules for converting text in the source language to the target language. The rules are often developed by linguists and may operate at the lexical, syntactic, or semantic level. This focus on rules gives the name to this area of study: Rule-based Machine Translation, or RBMT.

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

Statistical Machine Translation-

Statistical machine translation, or SMT for short, is the use of statistical models that learn to translate text from a source language to a target language.

Given a sentence T in the target language, we seek the sentence S from which the translator produced T. We know that our chance of error is minimized by choosing that sentence S that is most probable given T. Thus, we wish to choose S so as to maximize Pr(S|T).

The approach is data-driven, requiring only a corpus of examples with both source and target language text. This means linguists are no longer required to specify the rules of translation.

Neural Machine Translation-

Neural machine translation, or NMT for short, is the use of neural network models to learn a statistical model for machine translation.

The key benefit to the approach is that a single system can be trained directly on source and target text, no longer requiring the pipeline of specialized systems used in statistical machine learning.

Unlike the traditional phrase-based translation system which consists of many small sub-components that are tuned separately, neural machine translation attempts to build and train a single, large neural network that reads a sentence and outputs a correct translation.

Encoder-Decoder Model

Multilayer Perceptron neural network models can be used for machine translation, although the models are limited by a fixed-length input sequence where the output must be the same length.

These early models have been greatly improved upon recently through the use of recurrent neural networks organized into an encoder-decoder architecture that allow for variable length input and output sequences.

An encoder neural network reads and encodes a source sentence into a fixed-length vector. A decoder then outputs a translation from the encoded vector. The whole encoder–decoder system, which consists of the encoder and the decoder for a language pair, is jointly trained to maximize the probability of a correct translation given a source sentence.

Stream Tech Notes

Encoder-Decoders with Attention

Although effective, the Encoder-Decoder architecture has problems with long sequences of text to be translated.

The problem stems from the fixed-length internal representation that must be used to decode each word in the output sequence.

The solution is the use of an attention mechanism that allows the model to learn where to place attention on the input sequence as each word of the output sequence is decoded.

The encoder-decoder recurrent neural network architecture with attention is currently the state-of-the-art on some benchmark problems for machine translation. And this architecture is used in the heart of the Google Neural Machine Translation system, or GNMT, used in their Google Translate service.

Beam search and width

A machine translation model is similar to a language model except it has an encoder network placed before. For this reason, it is sometimes referred as a conditional language model.

Seq2seq(sequence to sequence) architectures are considered to be an important medium to MT(Machine Translation).This sequence comes under many to many sequence architecture of variable input and output length. Generally the architecture for MT consist of a encoder and a decoder. The encoder takes the embedding

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

RGPVNOTES.IN

of the words present in the vocabulary of one language, encodes it and provides it to a decoder as a starting activation. The decoder looks similar to language model which is used to generate random sequence, But the difference lies in here, language model is a one to many architectures which generate random sequences and the activation with which the model is initialized is 0. Whereas the decoder works on the theory of conditional probability and can be called as a Conditional Language model. It generates a sequence given an input i.e. the output of the encoder.

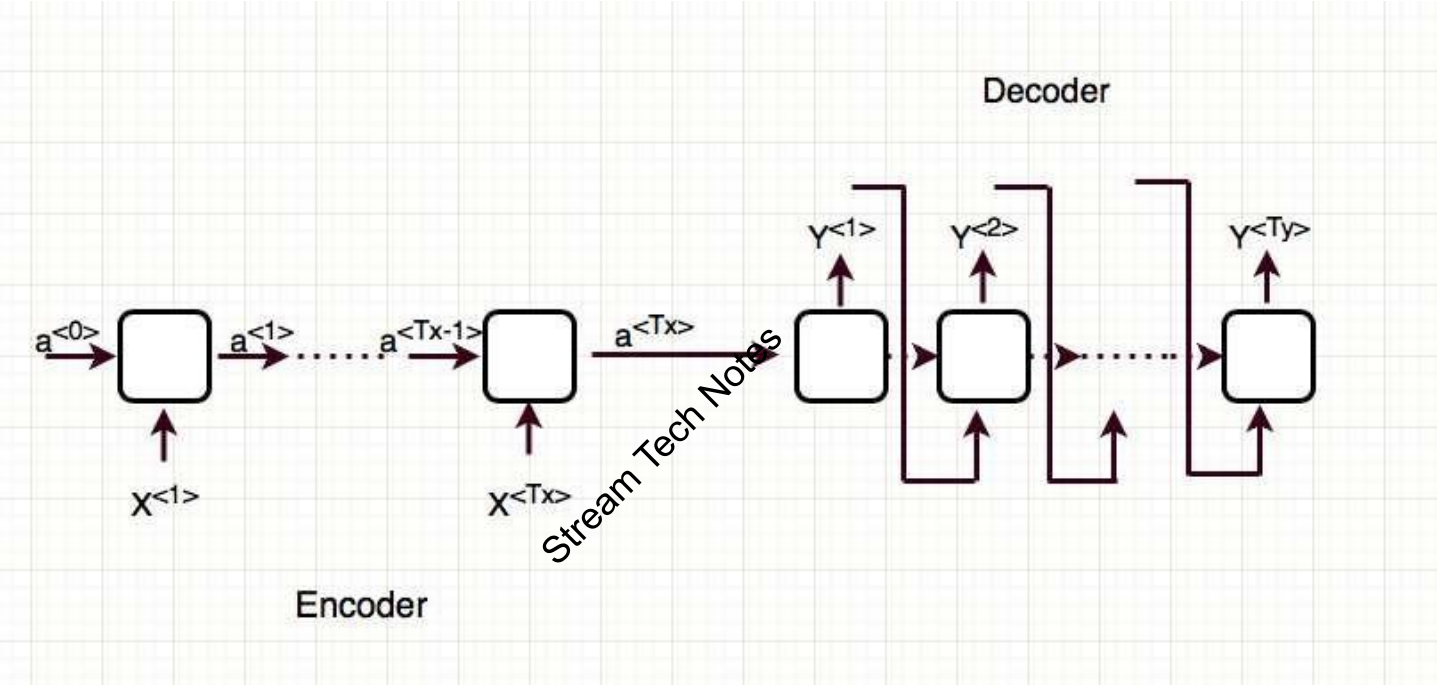


Figure: 4.8

Encoder and Decoder in Machine Translation

Given an input sequence x<1>, x<2>, x<3>,…., x<Tx> length Tx are in language-1 and the output generated by the decoder network y<1>, y<2>,…., y<Ty> be of length Ty in language-2. The output consist of a softmax layer. The outputs are given by the probability P(y<1>,y<2>,….., y<Ty>|a<Tx>). To pickup the most likely sentence this probability expression needs to be maximized i.e. :-

The goal is to find a sentence y such that:

$$y = \underset{y^{<1>},\ldots,y^{<T_y>}}{\arg\max}\ P(y^{<1>},\ldots,y^{<T_y>}|x)$$

One popular heuristic method to execute the purpose is Beam Search. Another solution to the above is the use of Greedy Search. Greedy Search takes only one output into account that reduces the possibilities to get other sentences also which can be more likely output to the translation.

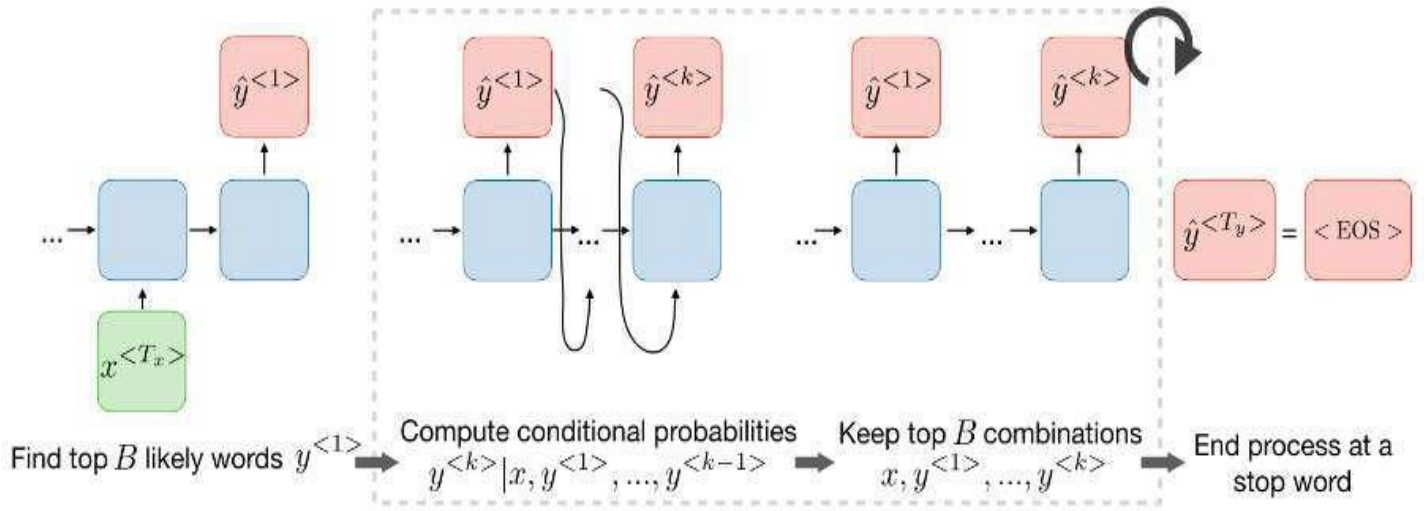
Beam search —

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

Beam search decoding iteratively creates text candidates (beams) and scores them.
It is a heuristic search algorithm used in machine translation and speech recognition to find the likeliest sentence y given an input x.

- Step 1: Find top B likely words $y^{<1>}$
- Step 2: Compute conditional probabilities $y^{<k>}|x, y^{<1>}, \ldots, y^{<k-1>}$
- Step 3: Keep top B combinations $x, y^{<1>}, \ldots, y^{<k>}$



Remark: if the beam width is set to 1, then this is equivalent to a naive greedy search.

Figure: 4.9

Beam width

The value of beam width for production purpose is generally kept between 10–100 and for research purpose this value is usually taken in between 1000 to 3000. More the beam width, more is the possibility of finding a likely sentence but it makes the computational expenses and memory requirement significantly high.
The beam width B is a parameter for beam search. Large values of B yield to better result but with slower performance and increased memory. Small values of B lead to worse results but is less computationally intensive. A standard value for B is around 10.

Length normalization — In order to improve numerical stability, beam search is usually applied on the following normalized objective, often called the normalized log-likelihood objective, defined as:

$$\text{Objective} = \frac{1}{T_y^{\alpha}} \sum_{t=1}^{T_y} \log \left[p(y^{<t>}|x, y^{<1>}, \ldots, y^{<t-1>}) \right]$$

Remark: the parameter αα can be seen as a softener, and its value is usually between 0.5 and 1.

Bleu score — The bilingual evaluation understudy (bleu) score quantifies how good a machine translation is by computing a similarity score based on n-gram precision. It is defined as follows:

$$\text{bleu score} = \exp\left(\frac{1}{n} \sum_{k=1}^{n} p_k \right)$$

where p_n is the bleu score on n-gram only defined as follows:

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

RGPVNOTES.IN

$$p_n = \frac{\sum_{\text{n-gram}\in\hat{y}} \text{count}_{\text{clip}}(\text{n-gram})}{\sum_{\text{n-gram}\in\hat{y}} \text{count}(\text{n-gram})}$$

Attention model

Attention was presented by Dzmitry Bahdanau, et al. in their paper "Neural Machine Translation by Jointly Learning to Align and Translate" that reads as a natural extension of their previous work on the Encoder-Decoder model.

Attention is proposed as a solution to the limitation of the Encoder-Decoder model encoding the input sequence to one fixed length vector from which to decode each output time step. This issue is believed to be more of a problem when decoding long sequences.

This model allows an RNN to pay attention to specific parts of the input that is considered as being important, which improves the performance of the resulting model in practice. By noting $\alpha^{<t,t'>}$ the amount of attention that the output $y^{<t>}$ should pay to the activation $a^{<t'>}$ and $c^{<t>}$ the context at time t, we have:

$$c^{<t>} = \sum_{t'} \alpha^{<t,t'>} a^{<t'>} \quad \text{with} \quad \sum_{t'} \alpha^{<t,t'>} = 1$$

Remark: the attention scores are commonly used in image captioning and machine translation.

Attention weight The amount of attention that the output $y^{<t>}$ should pay to the activation $a^{<t'>}$ is given by $\alpha^{<t,t'>}$ computed as follows:

$$\alpha^{<t,t'>} = \frac{\exp(e^{<t,t'>})}{\sum_{t''=1}^{T_x} \exp(e^{<t,t''>})}$$

Remark: computation complexity is quadratic with respect to Tx.

Reinforcement Learning

Reinforcement learning is a branch of machine learning that is concerned to take a sequence of actions in order to maximize some reward.

Basically an RL does not know anything about the environment, it learns what to do by exploring the environment. It uses actions, and receive states and rewards. The agent can only change your environment through actions.

One of the big difficulties of RL is that some actions take time to create a reward, and learning this dynamics can be challenging. Also the reward received by the environment is not related to the last action, but some action on the past.

Some concepts:

- Agents take actions in an environment and receive states and rewards
- Goal is to find a policy that maximize it's utility function
- Inspired by research on psychology and animal learning

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

RGPVNOTES.IN

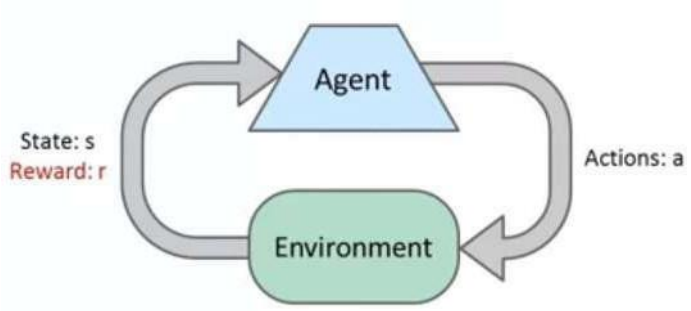


Figure: 4.10

Here we don't know which actions will produce rewards, also we don't know when an action will produce rewards, some times you do an action that will take time to produce rewards.

Basically all is learned with interactions with the environment.

Reinforcement learning components:

- Agent: Our robot
- Environment: The game, or where the agent lives.
- A set of states
- Policy: Map between state to actions
- Reward Function : Gives immediate reward for each state
- Value Function: Gives the total amount of reward the agent can expect from a particular state to all possible states from that state. With the value function you can find a policy.
- Model (Optional): Used to do planning, instead of simple trial-and-error approach common to Reinforcement learning. Here means the possible state after we do an action on the state

Stream Tech Notes

There is a variant of Reinforcement learning called Deep Reinforcement Learning where you use Neural Networks as function approximators for the following:

- Policy (Select next action when you are on some particular state)
- Value-Functions (Measure how good a state or state-action pair is right now)
- The whole Model/World dynamics, so you can predict next states and rewards.

MDP

MDP is a framework that can solve most Reinforcement Learning problems with discrete actions. With the Markov Decision Process, an agent can arrive at an optimal policy for maximum rewards over time.

The Markov decision process, better known as MDP, is an approach in reinforcement learning to take decisions in a grid world environment. A grid world environment consists of states in the form of grids.

The MDP tries to capture a world in the form of a grid by dividing it into states, actions, models/transition models, and rewards. The solution to an MDP is called a policy and the objective is to find the optimal policy for that MDP task.

Thus, any reinforcement learning task composed of a set of states, actions, and rewards that follows the Markov property would be considered an MDP.

The aim of MDP is to train an agent to find a policy that will return the maximum cumulative rewards from taking a series of actions in one or more states.

Here are the most important parts:

- States: A set of possible states
- Model: Probability to go to state when you do the action while you were on state , is also called transition model.

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

RGPVNOTES.IN

- Action: , things that you can do on a particular state
- Reward: , scalar value that you get for been on a state.
- Policy: , our goal, is a map that tells the optimal action for every state
- Optimal policy: , is a policy that maximize your expected reward

Bellman equations

The agent tries to get the most expected sum of rewards from every state it lands in. In order to achieve that we must try to get the optimal value function, i.e. the maximum sum of cumulative rewards. Bellman equation will help us to do so.

Using Bellman equation, the value function will be decomposed into two part; an immediate reward, ***R****t+1*, and discounted value of the successor state γ***V***(***S****t+1*),

$$v(s) = \mathbb{E}[G_t \mid S_t = s]$$

We unroll the return ***G****t,*

$$= \mathbb{E}\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s\right]$$
$$= \mathbb{E}\left[R_{t+1} + \gamma\left(R_{t+2} + \gamma R_{t+3} + \ldots\right) \mid S_t = s\right]$$

then substitute the return ***G****t+1*, starting from time step *t+1,*

$$= \mathbb{E}[R_{t+1} + \gamma G_{t+1} \mid S_t = s]$$

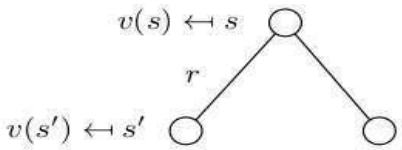
finally, since the expected value function is a linear function, meaning that ➪(a*X*+*bY*)= a➪(*X*) +b➪(*Y*). The expected value of the return ***G****t+1* is the value of the state ***S****t+1,*

$$= \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s]$$

That gives us the Bellman equation for MRPs,

$$v(s) = \mathbb{E}[R_{t+1} + \gamma v(S_{t+1}) \mid S_t = s]$$

So, for each state in the state space, the Bellman equation gives us the value of that state,



$$v(s) = \mathcal{R}_s + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'} v(s')$$

The value of the state ***S*** is the reward we get upon leaving that state, plus a discounted average over next possible successor states, where the value of each possible successor state is multiplied by the probability that we land in it.

Value Iteration and Policy Iteration

The value-iteration and policy-iteration algorithms are two fundamental methods for solving MDPs. Both value-iteration and policy-iteration assume that the agent knows the MDP model of the world (i.e. the agent knows

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

RGPVNOTES.IN

the state-transition and reward probability functions). Therefore, they can be used by the agent to (*offline*) plan its actions given knowledge about the environment before interacting with it.

Value iteration computes the optimal state value function by iteratively improving the estimate of **V(s)**. The algorithm initialize **V(s)** to arbitrary random values. It repeatedly updates the **Q(s, a)** and **V(s)** values until they converges. Value iteration is guaranteed to converge to the optimal values.

While value-iteration algorithm keeps improving the value function at each iteration until the value-function converges. Since the agent only cares about the finding the optimal policy, sometimes the optimal policy will converge before the value function. Therefore, another algorithm called policy-iteration instead of repeated improving the value-function estimate, it will re-define the policy at each step and compute the value according to this new policy until the policy converges. Policy iteration is also guaranteed to converge to the optimal policy and it often takes less iterations to converge than the value-iteration algorithm.

Value-Iteration vs Policy-Iteration

Both value-iteration and policy-iteration algorithms can be used for *offline planning* where the agent is assumed to have prior knowledge about the effects of its actions on the environment (they assume the MDP model is known). Comparing to each other, policy-iteration is computationally efficient as it often takes considerably fewer number of iterations to converge although each iteration is more computationally expensive.

Actor-critic model

1. The "Critic" estimates the value function. This could be the action-value (the *Q value*) or state-value (the *V value*).
2. The "Actor" updates the policy distribution in the direction suggested by the Critic (such as with policy gradients).

StreamTech Notes

And both the Critic and Actor functions are parameterized with neural networks.

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.

The actor takes as input the state and outputs the best action. It essentially controls how the agent behaves by learning the optimal policy (policy-based). The critic, on the other hand, evaluates the action by computing the value function (value based). Those two models participate in a game where they both get better in their own role as the time passes. The result is that the overall architecture will learn to play the game more efficiently than the two methods separately.

How Actor Critic works

Imagine you play a video game with a friend that provides you some feedback. You're the Actor and your friend is the Critic.

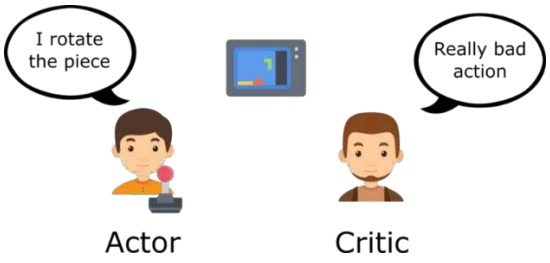


Figure: 4.11

At the beginning, you don't know how to play, so you try some action randomly. The Critic observes your action and provides feedback.

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

RGPVNOTES.IN

Learning from this feedback, you'll update your policy and be better at playing that game.
On the other hand, your friend (Critic) will also update their own way to provide feedback so it can be better next time.
The idea of Actor Critic is to have two neural networks. We estimate both:

$\pi(s, a, \theta)$ ACTOR : A policy function, controls how our agent acts.	$\hat{q}(s, a, w)$ CRITIC : A value function, measures how good these actions are.

Both run in parallel. Because we have two models (Actor and Critic) that must be trained, it means that we have two set of weights that must be optimized separately.

Q-learning

In the case where the agent does not know apriori what are the effects of its actions on the environment (state transition and reward models are not known). The agent only knows what are the set of possible states and actions, and can observe the environment current state. In this case, the agent has to actively learn through the experience of interactions with the environment. There are two categories of learning algorithms:

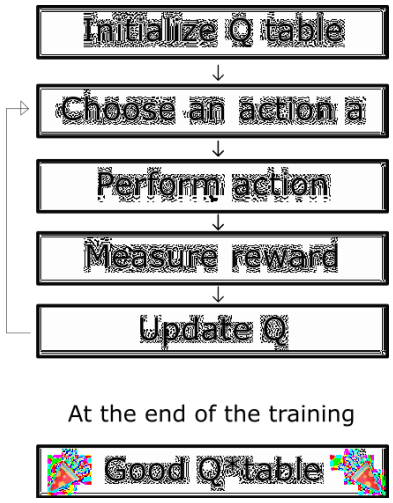
model-based learning: In model-based learning, the agent will interact to the environment and from the history of its interactions, the agent will try to approximate the environment state transition and reward models. Afterwards, given the models it learnt, the agent can use value-iteration or policy-iteration to find an optimal policy.

model-free learning: in model-free learning, the agent will not try to learn explicit models of the environment state transition and reward functions. However, it directly derives an optimal policy from the interactions with the environment.

StreamTech Notes

Q-Learning is an example of model-free learning algorithm. It does not assume that agent knows anything about the state-transition and reward models. However, the agent will discover what are the good and bad actions by trial and error.
The basic idea of Q-Learning is to approximate the state-action pairs Q-function from the samples of Q(s, a) that we observe during interaction with the environment. This approach is known as Time-Difference Learning.
Q-learning is an off policy reinforcement learning algorithm that seeks to find the best action to take given the current state. It's considered off-policy because the q-learning function learns from actions that are outside the current policy, like taking random actions, and therefore a policy isn't needed. More specifically, q-learning seeks to learn a policy that maximizes the total reward. The 'q' in q-learning stands for quality. Quality in this case represents how useful a given action is in gaining some future reward.

The Q-learning algorithm Process



SARSA

The SARSA stands for **State Action Reward State Action** which symbolizes the tuple (s, a, r, s', a') is an On-Policy

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

RGPVNOTES.IN

algorithm for TD-Learning. The major difference between it and Q-Learning, is that the maximum reward for the next state is not necessarily used for updating the Q-values. Instead, a new action, and therefore reward, is selected using the same policy that determined the original action. The name Sarsa actually comes from the fact that the updates are done using the quintuple **Q(s, a, r, s', a').** Where: **s, a** are the original state and action, **r** is the reward observed in the following state and **s', a'** are the new state-action pair.

SARSA vs Q-learning

The difference between these two algorithms is that **SARSA** chooses an action following the same current policy and updates its Q-values whereas **Q-learning** chooses the greedy action, that is, the action that gives the maximum Q-value for the state, that is, it follows an optimal policy.

Stream Tech Notes

RGPVNOTES.IN

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

RGPVNOTES.IN

Machine Learning (CS-601)

Class Notes

Unit V

Support Vector Machines

A support vector machine (SVM) is a supervised machine learning model that uses classification algorithms for two-group classification problems. After giving an SVM model set of labeled training data for either of two categories, they're able to categorize new examples.

Support Vector Machines is a fast and dependable classification algorithm that performs very well with a limited amount of data.

In machine learning, support-vector machines (SVMs), also support-vector networks 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.

Stream Tech Notes

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.

SVM algorithm can be used for **Face detection, image classification, text categorization,** etc.

Types of SVM

SVM can be of two types:

- **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
- **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

RGPVNOTES.IN

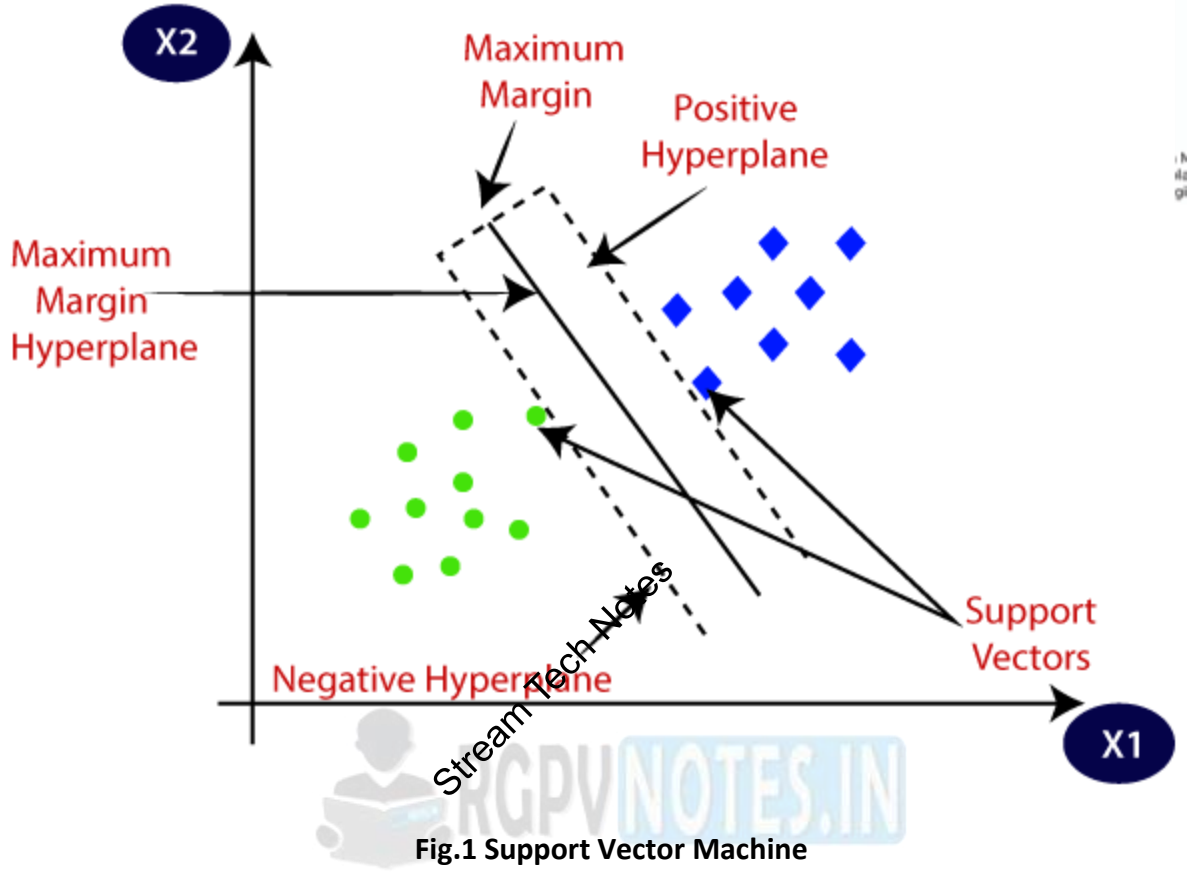


Fig.1 Support Vector Machine

Bayesian Learning

Bayesian machine learning is a particular set of approaches to probabilistic machine learning.

Bayesian learning treats model parameters as random variables - in Bayesian learning, parameter estimation amounts to computing posterior distributions for these random variables based on the observed data.

Bayesian learning typically involves generative models - one notable exception is **Bayesian linear regression**, which is a discriminative model.

Bayesian models

Bayesian modeling treats those two problems as one.

We first have a prior distribution over our parameters (i.e. what are the likely parameters?) P (θ).

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

RGPVNOTES.IN

From this we compute a posterior distribution which combines both inference and learning:

P(y1,…,yn,θ|x1,…,xn)=P(x1,…,xn,y1,…,yn|θ)P(θ)P(x1,…,xn)

Then prediction is to compute the conditional distribution of the new data point given our observed data, which is the marginal of the latent variables and the parameters:

P(xn+1|x1,…,xn)=∫P(xn+1|θ)P(θ|x1,…,xn)dθ

Classification then is to predict the distributions of the new datapoint given data from other classes, then finding the class which maximizes it:

P(xn+1|xc1,…,xcn)=∫P(xn+1|θc)P(θc|xc1,…,xcn)dθc

Naive Bayes

The main assumption of Naive Bayes is that all features are independent effects of the label. This is a really strong simplifying assumption but nevertheless in many cases Naive Bayes performs well.

Stream Tech Notes

Naive Bayes is also statistically efficient which means that it doesn't need a whole lot of data to learn what it needs to learn.

RGPVNOTES.IN

If we were to draw it out as a Bayes' net:

YYY→F1→F2…→Fn

Where Y is the label and F1,F2,…,Fn are the features.

The model is simply:

P(Y|F1,…,Fn) ϙ p(Y)∏iP(Fi|Y)

This just comes from the Bayes' net described above.

The Naive Bayes learns $P(Y,f_1,f_2,\ldots,f_n)$ which we can normalize (divide by $P(f_1,\ldots,f_n)$) to get the conditional probability $P(Y|f_1,\ldots,f_n)$:

$$P(y_1,f_1,\ldots,f_n) = \begin{matrix} P(Y,f_1,\ldots,f_n)\ P(y_1)\prod_i P(f_i|y_1) \\ P(y_2,f_1,\ldots,f_n)\ P(y2)\prod_i P(fi|y2) \\ P(y_k,f_1,\ldots,f_n)\ \ P(y_k)\prod_i P(f_i|y_k) \end{matrix}$$

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

RGPVNOTES.IN

So the parameters of Naive Bayes are P(Y) and P(Fi|Y) for each feature.

Inference in Bayesian models

Maximum a posteriori (MAP) estimation

A Bayesian alternative to MLE, we can estimate probabilities using *maximum a posteriori estimation*, where we instead choose a probability (a point estimate) that is most likely given the observed data:

$$\pi_{MAP} = \text{argmax}_{\pi} P(\pi|X)$$

$$= \text{argmax}_{\pi} \{P(X|\pi)P(\pi)\}/P(X)$$

$$= \text{argmax}_{\pi} P(X|\pi)P(\pi)$$

$$P(y|X) \approx P(y|\pi_{MAP})$$

So unlike MLE, MAP estimation uses Bayes' Rule so the estimate can use prior knowledge (P(π)) about what we expect π to be.

Stream Tech Notes

Again, this may be done with log-likelihoods:

$$\theta_{MAP} = \text{argmax}_{\theta} p(\theta|x) = \text{argmax}_{\theta} \log p(x|\theta) + \log p(\theta)$$

Maximum A Posteriori (MAP)

Likelihood function L(θ) is the probability of the data D as a function of the parameters θ.

This often has very small values so typically we work with the log-likelihood function instead:

ℓ(θ)=logL(θ)

The *maximum likelihood criterion* simply involves choosing the parameter θ to maximize ℓ(θ).

This can (sometimes) be done analytically by computing the derivative and setting it to zero and yields the *maximum likelihood estimate*.

MLE's weakness is that if you have only a little training data, it can overfit. This problem is known as data sparsity. For example, you flip a coin twice and it happens to land on heads both times. Your maximum likelihood estimate for θ (probability that the coin lands on heads) would be 1! We can then try to generalize this estimate to another dataset and test it by measuring the log-likelihood on the test set. If a tails shows up at all in the test set, we will have a test log-likelihood of −∞.

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

RGPVNOTES.IN

We can instead use Bayesian techniques for parameter estimation. In Bayesian parameter estimation, we treat the parameters θ as a random variable as well, so we learn a joint distribution p(θ,D).

We first require a prior distribution p(θ) and the likelihood p(D|θ)(as with maximum likelihood).

We want to compute p(θ|D), which is accomplished using Bayes' rule:

$$p(\theta|D)=p(\theta)p(D|\theta)/\{\int p(\theta')p(D|\theta')d\theta'\}$$

Though we work with only the numerator for as long as possible (i.e. we delay normalization until it's necessary) :

$$p(\theta|D)\propto p(\theta)p(D|\theta)$$

The more data we observe, the less uncertainty there is around the parameter, and the likelihood term comes to dominate the prior - we say that the *data overwhelm the prior*.

We also have the *posterior predictive distribution* p(D'|D), which is the distribution over future observables given past observations. This is computed by computing the posterior over θ and then marginalizing out θ:

Stream Tech Notes

$$p(D'|D)=\int p(\theta|D)p(D'|\theta)d\theta$$

The normalization step is often the most difficult, since we must compute an integral over potentially many, many parameters.

We can instead formulate Bayesian learning as an optimization problem, allowing us to avoid this integral. In particular, we can use *maximum a-posteriori* (MAP) approximation.

Whereas with the previous Bayesian approach (the "full Bayesian" approach) we learn a distribution over θ , with MAP approximation we simply get a point estimate (that is, a single value rather than a full distribution). In particular, we get the parameters that are most likely under the posterior:

$$\Theta\text{^}MAP=argmax_\theta p(\theta|D)$$

$$=argmax_\theta p(\theta,D)$$

$$=argmax_\theta p(\theta)p(D|\theta)$$

$$=argmax_\theta \log p(\theta)+\log p(D|\theta)$$

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

RGPVNOTES.IN

Maximizing $logp(D|\theta)$ is equivalent to MLE, but now we have an additional prior term $logp(\theta)$. This prior term functions somewhat like a regularizer. In fact, if p(θ) is a Gaussian distribution centered at 0, we have L2 regularization.

Application of machine learning in computer vision

Machine Learning in Computer Vision

It targets different application domains to solve critical real-life problems basing its algorithm from the human biological vision.

What is Computer Vision?

Computer vision is the process of understanding digital images and videos using computers. It seeks to automate tasks that human vision can achieve. This involves methods of acquiring, processing, analyzing, and understanding digital images, and extraction of data from the real world to produce information. It also has sub-domains such as object recognition, video tracking, and motion estimation, thus having applications in medicine, navigation, and object modeling.

Stream Tech Notes

To put it simply, computer vision works with a device using a camera to take pictures or videos, then perform analysis. The goal of computer vision is to understand the content of digital images and videos. Furthermore, extract something useful and meaningful from these images and videos to solve varied problems. Such examples are systems that can check if there is any food inside the refrigerator, checking the health status of ornamental plants, and complex processes such as disaster retrieval operation.

Tasks involving Computer Vision

Recognition in Computer Vision

- **Object recognition** – it involves finding and identifying objects in a digital image or video. It is most commonly applied in face detection and recognition. Object recognition can be approached through the use of either machine learning or deep learning.
- **Machine learning approach –** object recognition using machine learning requires the features to be defined first before being classified. A common approach that uses machine learning is the scale-invariant feature transform (SIFT). SIFT uses key points of objects and stores them in a database. When categorizing an image, SIFT checks the key points of the image, which matches those found in the database.
- **Deep learning approach –** object recognition using deep learning does not need specifically defined features. The common approaches that use deep learning are based on convolutional neural networks. A convolutional neural network is a type of deep neural network which is an artificial neural network with multiple layers between the input and output. An artificial neural network is a computing system inspired by the

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

RGPVNOTES.IN

biological neural network in the brain. The best example of this is the **ImageNet**. It is a visual database designed for object recognition in which the performance is said to be almost similar to that of humans.

- **Motion Analysis-** Motion Analysis in computer vision involves a digital video that is processed to produce information. Simple processing can detect the motion of an object. More complex processing tracks an object over time and can determine the direction of the motion. It has applications in motion capture, sports, and gait analysis.
- **Motion capture** – involves recording the movement of objects. Markers are worn near joints to identify motion. It has applications in animation, sports, computer vision, and gait analysis. Typically, only the movements of the actors are recorded and the visual appearance is not included.
- **Gait analysis –** is the study of locomotion and the activity of muscles using instruments. It involves quantifying and interpreting the gait pattern. Several cameras linked to a computer are required. The subject wears markers at various reference points of the body. As the subject moves, the computer calculates the trajectory of each marker in three dimensions. It can be applied to sports biomechanics.
- **Video tracking** – is a process of locating a moving object over time. Object recognition is used to aid in video tracking. Video tracking can be used in sports. Sports involve a lot of movement, and these technologies are ideal for tracking the movement of players.
- **Autonomous vehicles** – computer vision is used in autonomous vehicles such as a self-driving car. Cameras are placed on top of the car providing 360 degrees field of vision up to 250 meters of range. The cameras aid in lane finding, road curvature estimation, obstacle detection, traffic sign detection, and many more. Computer vision has to implement object detection and classification.
- **Sports –** computer vision is used in sports to improve the broadcast experience, athlete training, analysis and interpretation, and decision making. Sports biomechanics is a quantitative study and analysis of athletes and sports. For broadcast improvement, virtual markers can be drawn across the field or court. As for athlete training, creating a skeleton model of an acrobat and estimating the center of mass allows for improvement in form and posture. Finally, for sports analysis and interpretation, players are tracked in live games allowing for real-time information.

Stream Tech Notes

Computer vision is used to acquire the data to achieve basketball analytics. These analytics are retrieved using video tracking and object recognition by tracking the movement of the players. Motion analysis methods are also used to assist in motion tracking. Deep learning using convolutional neural networks is used to analyze the data.

Sub-domains of computer vision include scene reconstruction, event detection, video tracking, object recognition, 3D pose estimation, learning, indexing, motion estimation, and image restoration.

Computer Vision, often abbreviated as CV, is defined as a field of study that seeks to develop techniques to help computers “see” and understand the content of digital images such as

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

RGPVNOTES.IN

photographs and videos.The goal of the field of computer vision and its distinctness from image processing.

Application of Machine Learning in Speech Processing

Speech Recognition

Speech recognition (SR) is the translation of spoken words into text. It is also known as "automatic speech recognition" (ASR), "computer speech recognition", or "speech to text" (STT).

In speech recognition, a software application recognizes spoken words. The measurements in this application might be a set of numbers that represent the speech signal. We can segment the signal into portions that contain distinct words or phonemes. In each segment, we can represent the speech signal by the intensities or energy in different time-frequency bands.

Speech recognition applications include voice user interfaces. Voice user interfaces are such as voice dialing, call routing, domestic appliance control. It can also use as simple data entry, preparation of structured documents, speech-to-text processing, and plane.

Stream Tech Notes

Using Machine Learning, *Baidu's* research and development department have created a tool called **Deep Voice** – a deep neural network that is capable of producing artificial voices that are difficult to distinguish from real human voice. This network can "*learn*" features in rhythm, voice, pronunciation, and vocalization to create the voice of the speaker. In addition, Google also uses Machine Learning for other voice-related products and translations such as Google Translate, Google Text To Speech, Google Assistant.

Dynamic time warping (DTW)-based speech recognition

Dynamic time warping is an algorithm for measuring similarity between two sequences that may vary in time or speed.

Applications of Machine Learning in Natural Language Processing

Machine learning for natural language processing and text analytics involves using machine learning algorithms and "narrow" artificial intelligence (AI) to understand the meaning of text documents. These documents can be just about anything that contains text: social media comments, online reviews, survey responses, even financial, medical, legal and regulatory documents. In essence, the role of machine learning and AI in natural language processing (NLP) and text analytics is to improve, accelerate and automate the underlying text analytics functions and NLP features that turn this unstructured text into useable data and insights.

Machine learning for NLP and text analytics involves a set of statistical techniques for identifying parts of speech, entities, sentiment, and other aspects of text. The techniques can

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

RGPVNOTES.IN

be expressed as a model that is then applied to other text, also known as supervised machine learning. It also could be a set of algorithms that work across large sets of data to extract meaning, which is known as unsupervised machine learning.

Supervised Machine Learning for Natural Language Processing and Text Analytics

In supervised machine learning, a batch of text documents are tagged or annotated with examples of what the machine should look for and how it should interpret that aspect. These documents are used to "train" a statistical model, which is then given un-tagged text to analyze.

The most popular supervised NLP machine learning algorithms are:

- Support Vector Machines
- Bayesian Networks
- Maximum Entropy
- Conditional Random Field
- Neural Networks/Deep Learning

Stream Tech Notes

Tokenization

Tokenization involves breaking a text document into pieces that a machine can understand, such as words.

RGPVNOTES.IN

Part of Speech Tagging

Part of Speech Tagging (PoS tagging) means identifying each token's part of speech (noun, adverb, adjective, etc.) and then tagging it as such. PoS tagging forms the basis of a number of important Natural Language Processing tasks. We need to correctly identify Parts of Speech in order to recognize entities, extract themes, and to process sentiment. ***Lexalytics*** has a highly-robust model that can PoS tag with >90% accuracy, even for short, social media posts.

Named Entity Recognition

At their simplest, named entities are people, places, and things (products) mentioned in a text document. Unfortunately, entities can also be hashtags, emails, mailing addresses, phone numbers, and Twitter handles. In fact, just about anything can be an entity if you look at it the right way. And don't get us stated on tangential references.

Sentiment Analysis

Sentiment analysis is the process of determining whether a piece of writing is positive, negative or neutral, and then assigning a weighted sentiment score to each entity, theme, topic, and category within the document. This is an incredibly complex task that varies wildly with context.

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

RGPVNOTES.IN

For example, take the phrase, “sick burn” In the context of video games, this might actually be a positive statement.

Unsupervised Machine Learning for Natural Language Processing and Text Analytics

Unsupervised machine learning involves training a model without pre-tagging or annotating.

Matrix Factorization is another technique for unsupervised NLP machine learning. This uses “latent factors” to break a large matrix down into the combination of two smaller matrices. Latent factors are similarities between the items.

Another type of unsupervised learning is **Latent Semantic Indexing** (LSI). This technique identifies on words and phrases that frequently occur with each other. Data scientists use LSI for faceted searches, or for returning search results that aren’t the exact search term.

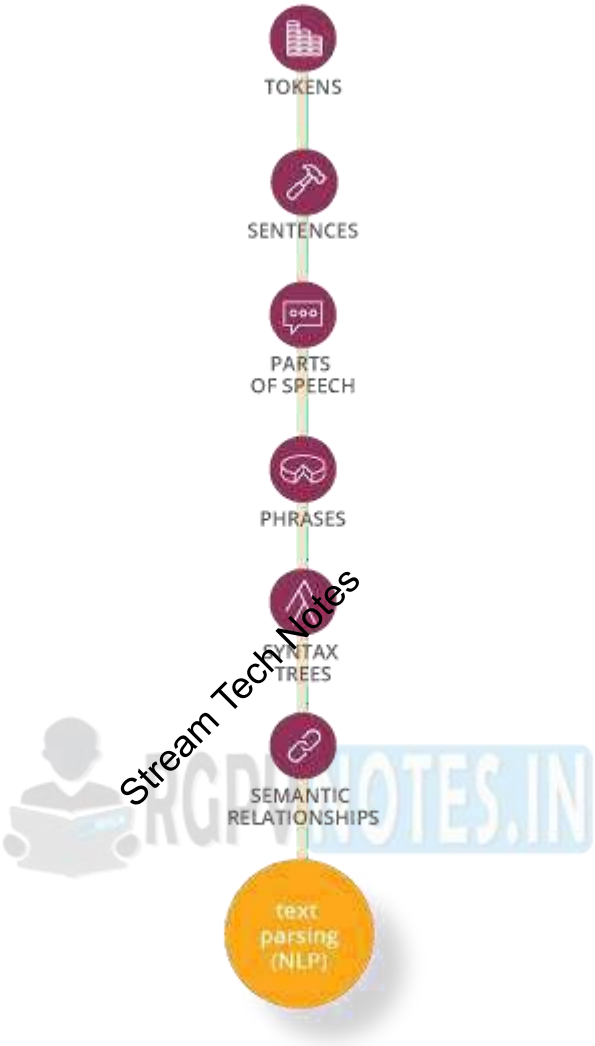
Stream Tech Notes

RGPVNOTES.IN

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

RGPVNOTES.IN



Stream Tech Notes

Fig. 2 Machine Learning Systems for NLP

Low-level text functions are the initial processes through which you run any text input. These functions are the first step in turning unstructured text into structured data; thus, these low-level functions form the base layer of information from which our mid-level functions draw on. Mid-level text analytics functions involve extracting the real content of a document of text. This means who is speaking, what they are saying, and what they are talking about.

The high-level function of sentiment analysis is the last step, determining and applying sentiment on the entity, theme, and document levels.

Low-Level

- **Tokenization**: ML + Rules

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

RGPVNOTES.IN

- **PoS Tagging**: Machine Learning
- **Chunking**: Rules
- **Sentence Boundaries**: ML + Rules
- **Syntax Analysis**: ML + Rules

Mid-Level

- Entities**: ML + Rules to determine "Who, What, Where"**
- Themes**: Rules "What's the buzz?"**
- Topics**: ML + Rules "About this?"**
- Summaries**: Rules "Make it short"**
- Intentions**: ML + Rules "What are you going to do?"**
    - Intentions uses the syntax matrix to extract the intender, intendee, and intent
    - We use ML to train models for the different types of intent
    - We use rules to whitelist or blacklist certain words
    - Multilayered approach to get you the best accuracy

High-Level

- **Apply Sentiment**: ML + Rules "How do you feel about that?"

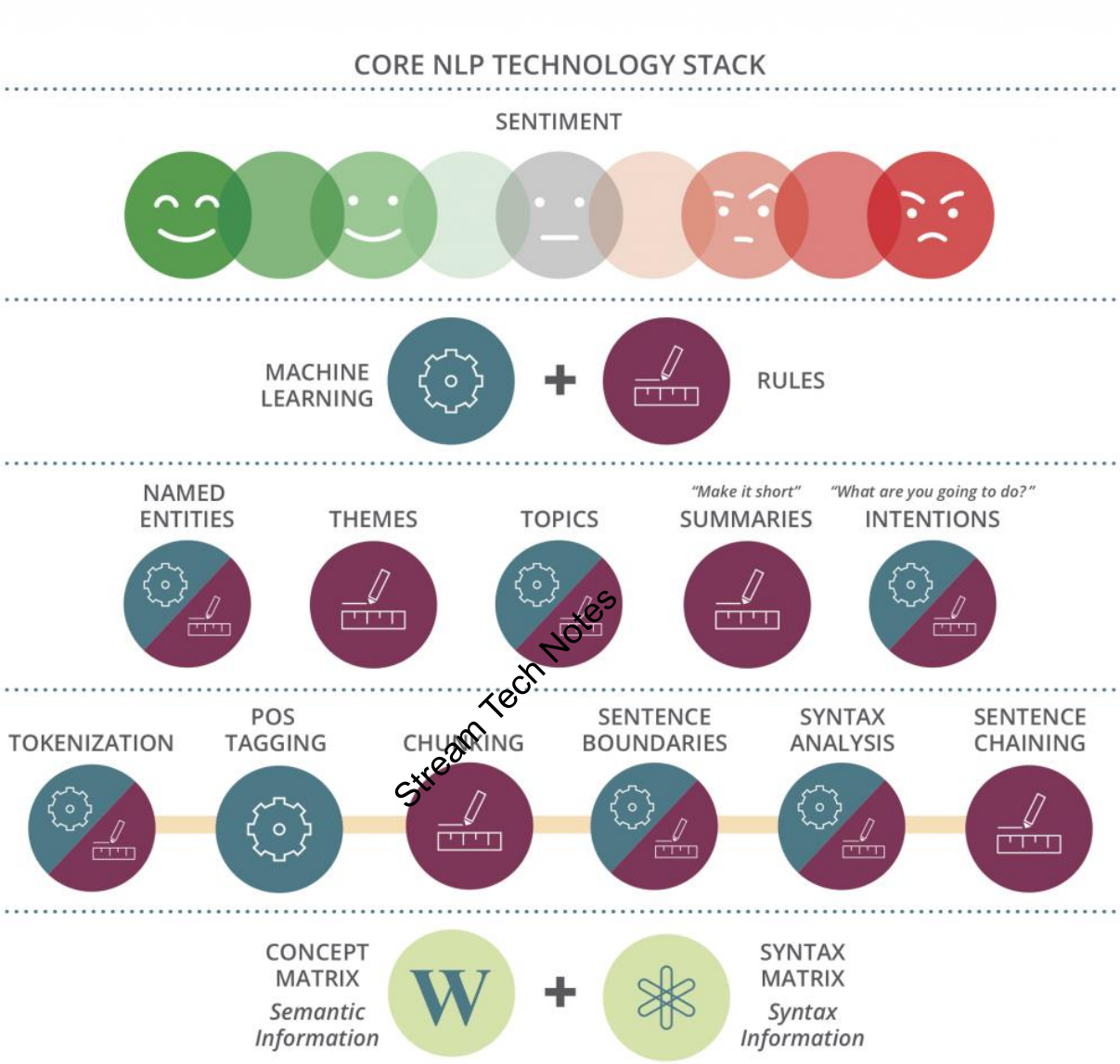
Stream Tech Notes

RGPVNOTES.IN

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

RGPVNOTES.IN



Stream Tech Notes

Fig. 3 Machine Learning in NLP

Case Study: ImageNet Competition

The recently proposed ImageNet dataset consists of several million images, each annotated with a single object category. However, these annotations may be imperfect, in the sense that many images contain multiple objects belonging to the label vocabulary. In other words, we have a multi-label problem but the annotations include only a single label (and not necessarily the most prominent).Such a setting motivates the use of a robust evaluation measure, which allows for a limited number of labels to be predicted and, as long as one of the predicted labels is correct, the overall prediction should be considered correct. This is indeed the type of evaluation measure used to assess algorithm performance in a recent competition on ImageNet data. Optimizing such types of performance measures presents several hurdles even with existing structured output learning methods. Indeed, many of the current state-of-the-art

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

RGPVNOTES.IN

methods optimize the prediction of only a single output label, ignoring this 'structure' altogether.

The recently proposed ImageNet project consists of building a growing dataset using an image taxonomy based on the **WordNet** hierarchy. Each node in this taxonomy includes a large set of images (in the hundreds or thousands). From an object recognition point of view, this dataset is interesting because it naturally suggests the possibility of leveraging the image taxonomy in order to improve recognition beyond what can be achieved independently for each image. Indeed this question has been the subject of much interest recently, culminating in a competition in this context using **ImageNet** data.

Although in ImageNet each image may have several objects from the label vocabulary, the annotation only includes a single label per image, and this label is not necessarily the most prominent. This imperfect annotation suggests that a meaningful performance measure in this dataset should somehow not penalize predictions that contain legitimate objects that are missing in the annotation. One way to deal with this issue is to enforce a robust performance measure based on the following idea: an algorithm is allowed to predict more than one label per image (up to a maximum of K labels), and as long as one of those labels agrees with the ground-truth label, no penalty is incurred. This is precisely the type of performance measure used to evaluate algorithm performance in the aforementioned competition.

Steam Tech Notes

The **ImageNet** project is a large visual database designed for use in visual object recognition software research. More than 14 million images have been hand-annotated by the project to indicate what objects are pictured and in at least one million of the images, bounding boxes are also provided. ImageNet contains more than 20,000 categories with a typical category, such as "balloon" or "strawberry", consisting of several hundred images. The database of annotations of third-party image URLs is freely available directly from ImageNet, though the actual images are not owned by ImageNet. Since 2010, the ImageNet project runs an annual software contest, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), where software programs compete to correctly classify and detect objects and scenes. The challenge uses a "trimmed" list of one thousand non-overlapping classes.

Significance for deep learning

On 30 September 2012, a convolutional neural network (CNN) called ***AlexNet*** achieved a top-5 error of 15.3% in the ImageNet 2012 Challenge, more than 10.8 percentage points lower than that of the runner up. This was made feasible due to the use of Graphics processing units (GPUs) during training, an essential ingredient of the deep learning revolution. According to The Economist, "Suddenly people started to pay attention, not just within the AI community but across the technology industry as a whole."

In 2015, **AlexNet** was outperformed by Microsoft's very deep CNN with over 100 layers, which won the ImageNet 2015 contest.

Get real-time updates from RGPV

Downloaded from www.rgpvnotes.in

RGPVNOTES.IN

Dataset

ImageNet crowdsources its annotation process. Image-level annotations indicate the presence or absence of an object class in an image, such as "there are tigers in this image" or "there are no tigers in this image". Object-level annotations provide a bounding box around the (visible part of the) indicated object. **ImageNet** uses a variant of the broad **WordNet** schema to categorize objects, augmented with 120 categories of dog breeds to showcase fine-grained classification. One downside of WordNet use is the categories may be more "elevated" than would be optimal for ImageNet: "Most people are more interested in Lady Gaga or the iPod Mini than in this rare kind of diplodocus." In 2012 ImageNet was the world's largest academic user of Mechanical Turk. The average worker identified 50 images per minute.

Stream Tech Notes

RGPVNOTES.IN

Get real-time updates from RGPV