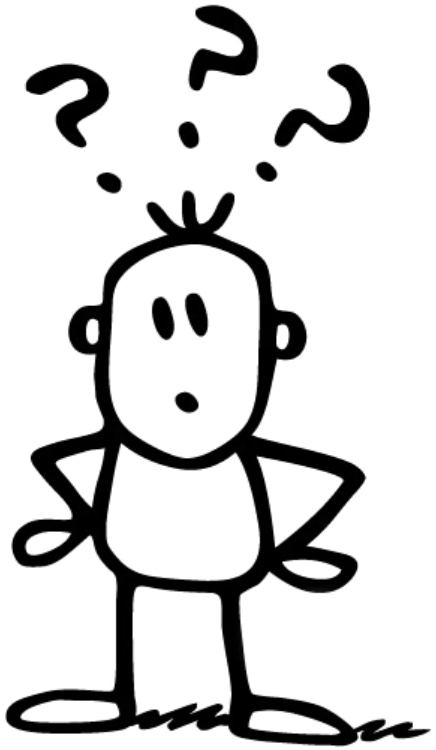
Intelligent Systems Design through Deep Learning

Shaik Nagur Shareef

Software Engineer

Outline

- Introduction to Artificial Intelligence
- Approaches to achieve AI
- Intuition to Machine Learning
- Introduction to Deep Learning
- Deep Learning Tools & Applications
- Feed Forward Neural Networks
    - Multi Layered Perceptron
    - Convolutional Neural Network
- Feedback Neural Networks
    - Recurrent Neural Networks (LSTM & GRU)
- Transfer Learning
- Transformers for Vision & Sequence Modeling



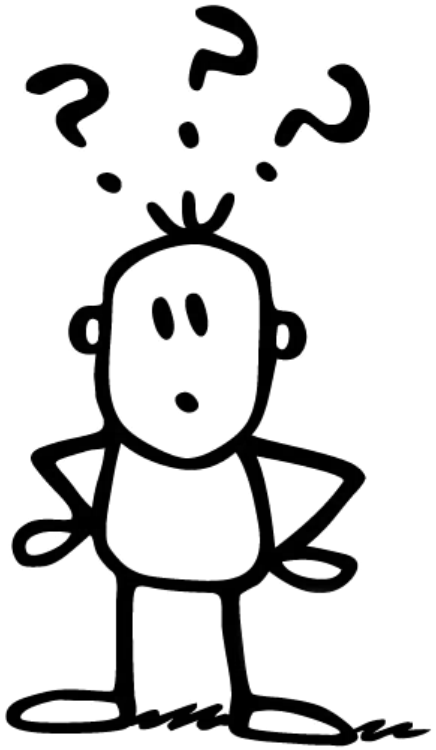
What is Intelligence..???

Intelligence…

Intelligence has been defined in many ways:

- The capacity for logic, understanding, self-awareness, learning, emotional knowledge, reasoning, planning, creativity, critical thinking, and problem-solving.

- More generally, it can be described as the ability to perceive or infer information, and to retain it as knowledge to be applied towards adaptive behaviours within an environment or context.



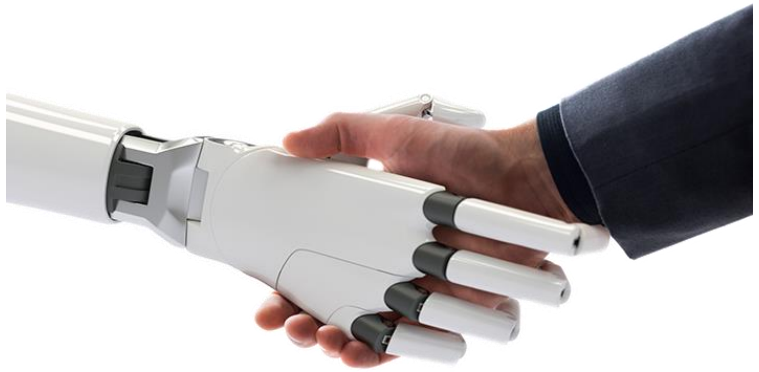
What is Artificial Intelligence.?

Artificial Intelligence…

- Artificial intelligence is the science and engineering of making Intelligent Systems
  - **Artificial:** Non Natural
  - **Intelligence:** Ability to learn, understand and think

- AI can be defined as Making the:
  - Systems that act & think like Humans
  - Systems that act & think Rationally

- ***Note:*** Rationally means doing right thing at right time



Fields of AI

- AI is broad **branch of computer science**.
- The goal of AI is to create systems, that can **function intelligently and independently**.
- Humans can speak and listen to communicate through language, in AI is a field of **speech recognition**.
- Humans can read and write a text in a language, in AI is a field of NLP or **Natural Language Process**.
- Humans can see with their eyes and process what they see, in AI is a **Computer Vision**. This falls under Symbolic Learning.

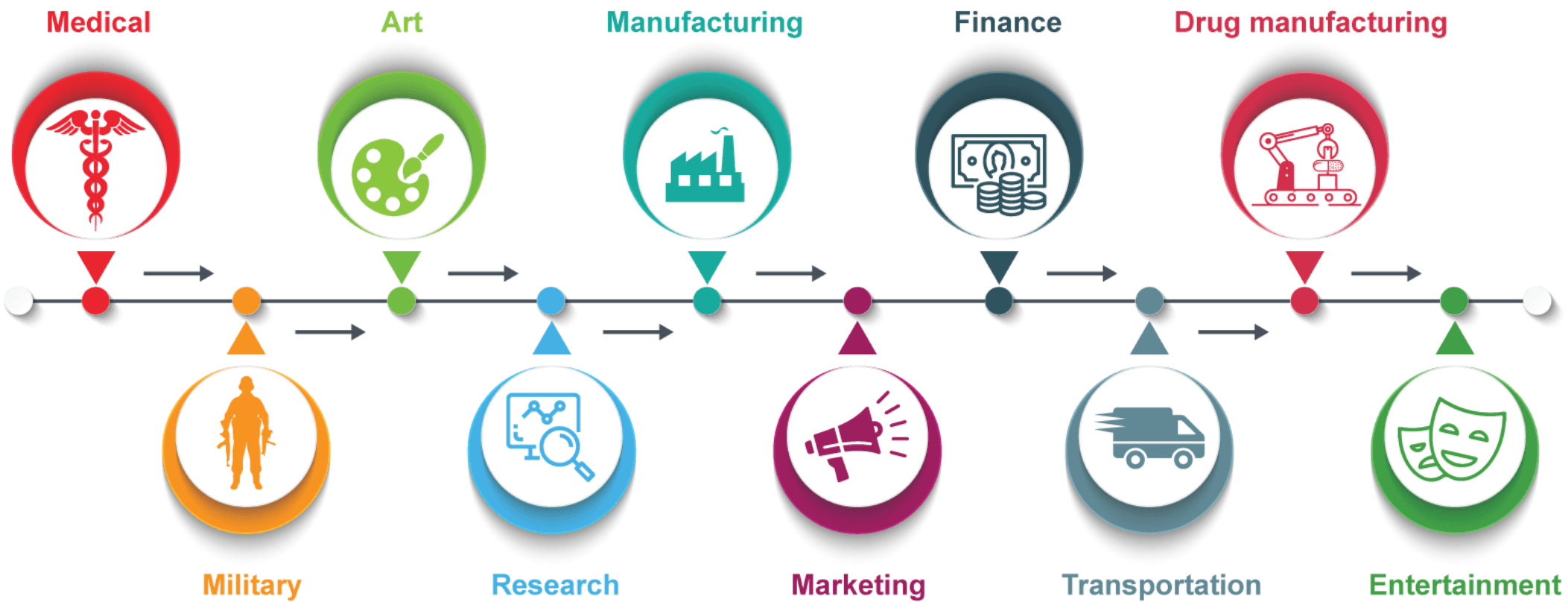
Fields of AI

- Human brain is **Network of Neurons** and we use this to learn things.
- If we replicate the structure and function of human brain, we might be able to get cognitive capabilities of machines. This is a field of **Neural Networks**.
- These networks are more complex and deeper and will use to learn complex things, that is a field of **Deep Learning**.
- There are different types of machine learnings, which are essentially use different techniques to replicate what human brain does.
- If a get a network to scan images from left to right and top to bottom it's a **Convolution Neuronal Network**.
- This CNN is used to recognize objects to seen, this is how **Computer Vision** fits in an **Object Recognition** is accomplished through AI.

Fields of AI

- Humans recognizes see around them, through eyes which creates images that world, in AI, **Image Processing** which is not directly not related to AI but required for field of Computer Vision.
- Humans can understand their environment and move around fluently, this is a field of **Robotics**.
- Humans has ability to see patterns such as grouping like objects, this is a field of **Patterns Recognition**.

Fields of Applications…



Techniques to achieve AI

Rules and Logic Based Approach

Machine Learning Based Approach

Rules and Logic Based Approach

Representing Process or System Using Logical Rules

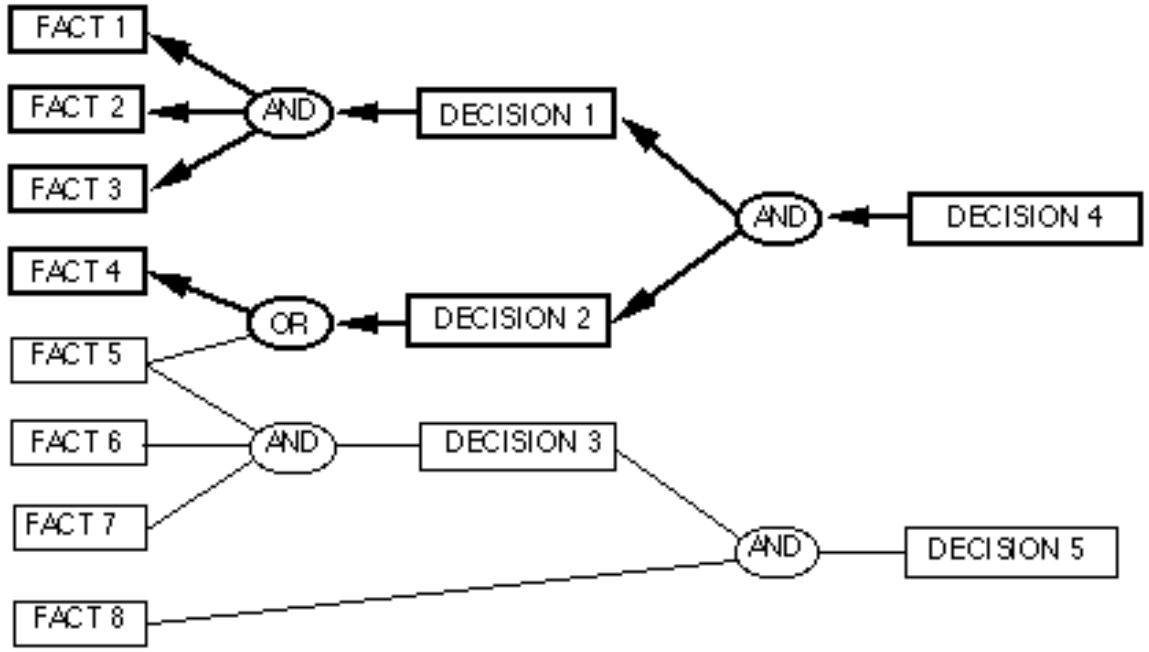
Top-down Rules are Created For Computer

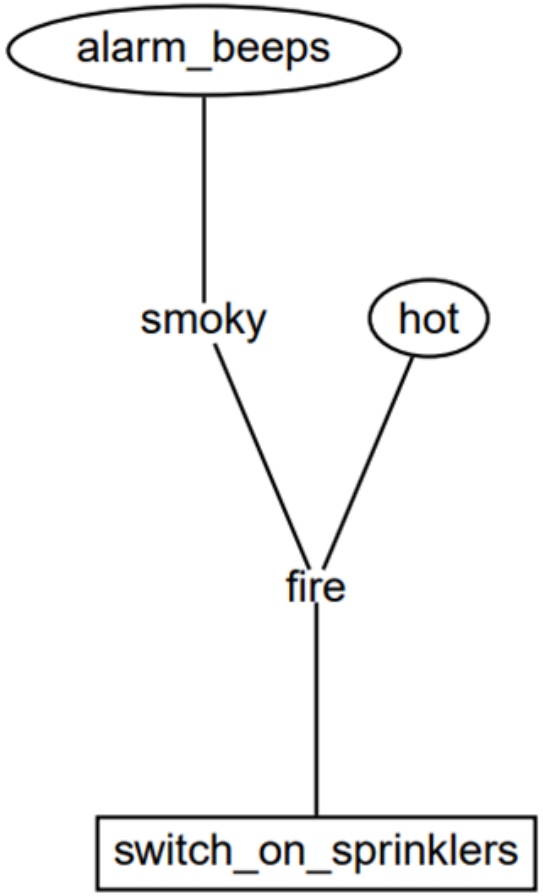
Computers Reason About These Rules

Can Be Used To Automate Process

E.G. Personal Income Tax Laws

Working of Rules and Logic Based Systems





Problems with **Logic and Rules-Based Approach**

- ✓It takes lot of time to list out all the rules.
- ✓Changing rules is tedious.
- ✓As the list of rules grows, it becomes too difficult to manage and has a lot of redundancies.
- ✓The person who wrote the initial rules for you leaves, and you have to spend time and resources to catch up on the long list of rules.

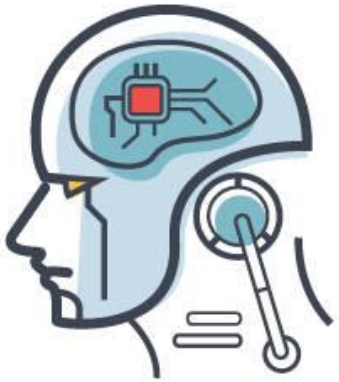
Machine Learning

- Algorithms find patterns in data and infer rules on their own
- "Learn" from data and improve over time
- These patterns can be used for automation or prediction
- ML is the **dominant** mode of AI today

Definition…

- Machine Learning is the field of study that **gives computers the ability to learn without being explicitly programmed**.

  - Arthur Samuel



- A ** (computer program) is said to learn from **experience E** with respect to some **class of tasks T** and **performance measure P**, if its performance at tasks in T, as measured by P, improves with experience E.

  - Tom Mitchell

Learning Systems

Task (T):

- Classification
- Regression
- Transcription
- Machine Translation
- Captioning
- Clustering
- Anomaly Detection
- De-noising

Performance Measure (P):

- Accuracy
- Precision
- Recall
- F1 Score
- Mean Squared Error
- BLUE
- Silhouette Score
- Similarity Measures

Experience (E):

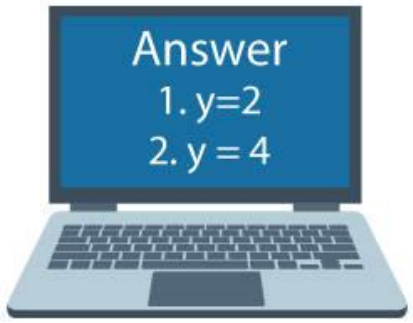
- Supervised
- Un-supervised
- Reinforced
- Semi-supervised

- Dataset

Machine Learning…

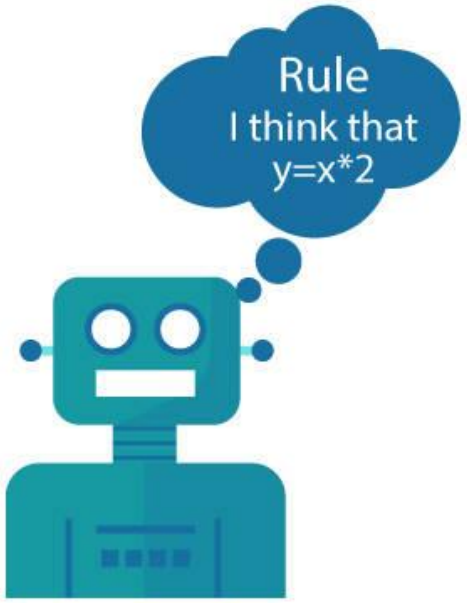
- Machine learning is a tool for turning information into knowledge.
- We are drowning in information and starving for knowledge.
  - **Google**: 24 petabytes of data are processed per day.
  - **Facebook**: 10 million photos are uploaded every hour.
  - **You Tube**: 1 hour of video is uploaded every second.
  - **Twitter**: 400 million tweets are posted per day.
- We are DATAFIED! Wherever we go, we leave a data trail. Data becomes fruitless unless we discover the hidden patterns.
- Wondering how? Yes! Machine Learning is a magic wand that turns information into knowledge, which will do wonders for humankind.

Traditional Learning



Data	Rule
1. x=1	y=x*2
2. x=2	y=x*2

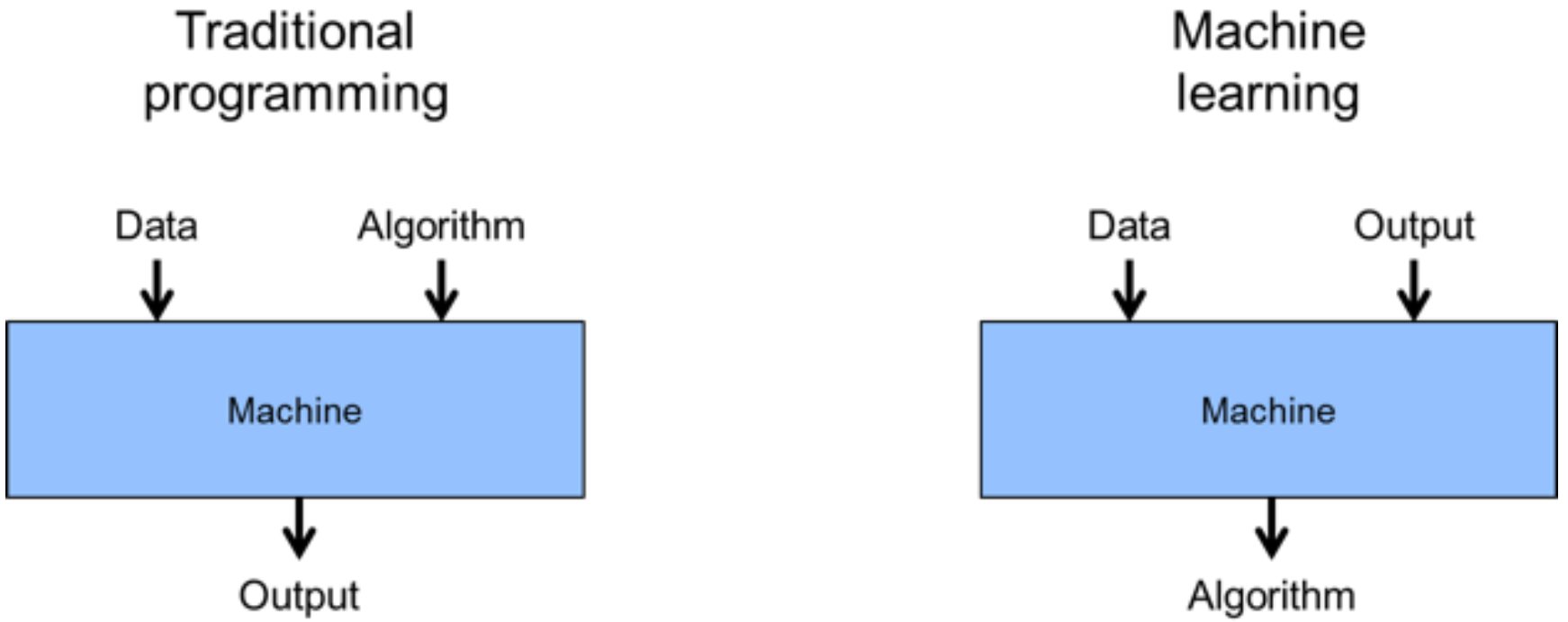
Machine Learning



Data	Answer
1. x=1	y=2
2. x=2	y=4

How Machine Learning is different from Traditional Learning…?

Traditional Programming vs Machine Learning



Why Machine Learning…?

- We interact with Machine Learning models every single day without our knowledge.
- Every time we perform a Google search, listen to a song or even take a photo, Machine Learning is becoming a backbone process behind it by invariably **learning and improving from every interaction**.
- Machines can drive your car for you, detect eye diseases, unlock your phone with face recognition, and the list never ends.

What is the life cycle of Machine Learning?

The Machine Learning process lies in 9 steps:

1. Defining Project Objectives
2. Gathering Data
3. Exploratory Data Analysis (EDA) and Data Cleaning
4. Choosing a Model
5. Training
6. Evaluation
7. Hyperparameter Tuning
8. Interpret and Communicate
9. Deployment and Documentation

Defining Project Objectives

- The first step of the life cycle is to recognize the opportunity for tangible improvement of activities, enhance customer satisfaction, or create value otherwise.
- It is critical that you understand the problem you are trying to solve. In this stage, you should also be identifying the central objectives of your project by identifying the variables that need to be predicted.

Gathering Data

- This is considered to be the primary step of the Machine Learning process.
- The quality and quantity of data you gather in this step will determine how efficient your model will be.

Some important things to remember while gathering data are:

- ✓Data can be collected from anywhere in any format.
- ✓More training examples will aid the model to be more efficient.
- ✓Make sure the number of samples for every class or topic is not overly imbalanced.
- ✓Ensure that your samples adequately cover the space of possible inputs, not only the common cases.

Exploratory Data Analysis (EDA) and Data Cleaning

Exploratory Data Analysis (EDA):

- Analyzing datasets to summarize their notable characteristics is called Exploratory Data Analysis.
- It helps in performing investigations on data to discover hidden patterns, anomalies, and so on.
- It aids in checking assumptions and hypothesis with the help of summary statistics.

Data Cleaning:

- Data can have several shortcomings. A few are:
    1. Missing values
    2. Duplicate data
    3. Invalid data
- The process of detecting, correcting, and ensuring that the given dataset is error-free, consistent enough to use, is called Data Cleaning.

Choosing a Model

- There are numerous models that researchers and Data scientists have created over the years.
- Some are very well-suited for image data, while others are suited for sequences, text-based data, and many more.
- Choosing the right model for the problem will impact the efficiency of the model.

Training

- The next step of the Machine Learning process, often known as the bulk of ML, is Training the model.
- This step is very similar to a person who is learning to drive for the first time. Though they do not know any of the basics initially, a licensed driver emerges eventually, after a lot of practice and feedback.
- The data is split into Training Data and Testing Data.
- The model is trained with the training data using different ML algorithms by adjusting the parameters in multiple iterations.
- The testing data is put aside as unseen data to evaluate your models.

Evaluation

- Once the training is complete, it is time to see if the model is any good, using Evaluation.
- This is where that dataset that we set aside earlier comes into play, that is, Testing Data.
- Evaluation allows us to test our model against the data that has never been used for training.
- This metric will enable us to see how the model might perform against data that it has not yet seen.
- This is meant to be representative of how the model might perform in the real world.

Hyperparameter Tuning

- After the evaluation step, it is time to see if we can further improve our training by tuning different parameters that were implicitly assumed in the training process. This process is called Hyperparameter Tuning.
- The tuned model is once again evaluated for model performance, and this cycle continues until the final best performing model is chosen.

Interpret and Communicate

- The most challenging task of the ML project is explaining the model's output.
- During the earlier days, Machine Learning was considered a BlackBox because it was hard to interpret their insights and values.
- The more interpretable your model is, the easier it is to communicate your model's importance to the stakeholders.

Deployment and Documentation

- Model deployment often poses a problem because of the coding and data science experience it requires and because the time-to-implementation of traditional data science methods from the start of the cycle is prohibitively long.
- The trained model has to be deployed in a real-world system to be efficient to humans.
- It can be deployed using any framework like Flask, Cloud, Azure, and so on.
- Document your project well for your successors to handle it.

Types of Machine Learning

- Machine Learning is an umbrella term that covers 3 learning techniques.

1. Supervised Learning
2. Unsupervised Learning
3. Reinforcement Learning

Supervised Learning

- Supervised learning is the Machine Learning task of learning a function that maps an input to an output based on example input-output pairs.
- It infers a function from labeled training data.
- Each training example is a pair consisting of an input object and the desired output value.
- A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples.

Applications:

1. Spam Detection
2. Pattern Recognition
3. Speech Recognition

Unsupervised Learning

- Unsupervised Learning helps in uncovering hidden patterns from unlabeled data.

Applications:

1. Recommender Systems
2. Targeted Marketing
3. Customer Segmentation
4. Structure Discovery

Reinforcement Learning

- **Reinforcement Learning** is a type of Machine Learning in which software agents ought to take actions in an environment to maximize the notion of cumulative reward.

- **Applications**
  1. Genetics
  2. Economics
  3. Robot Navigation

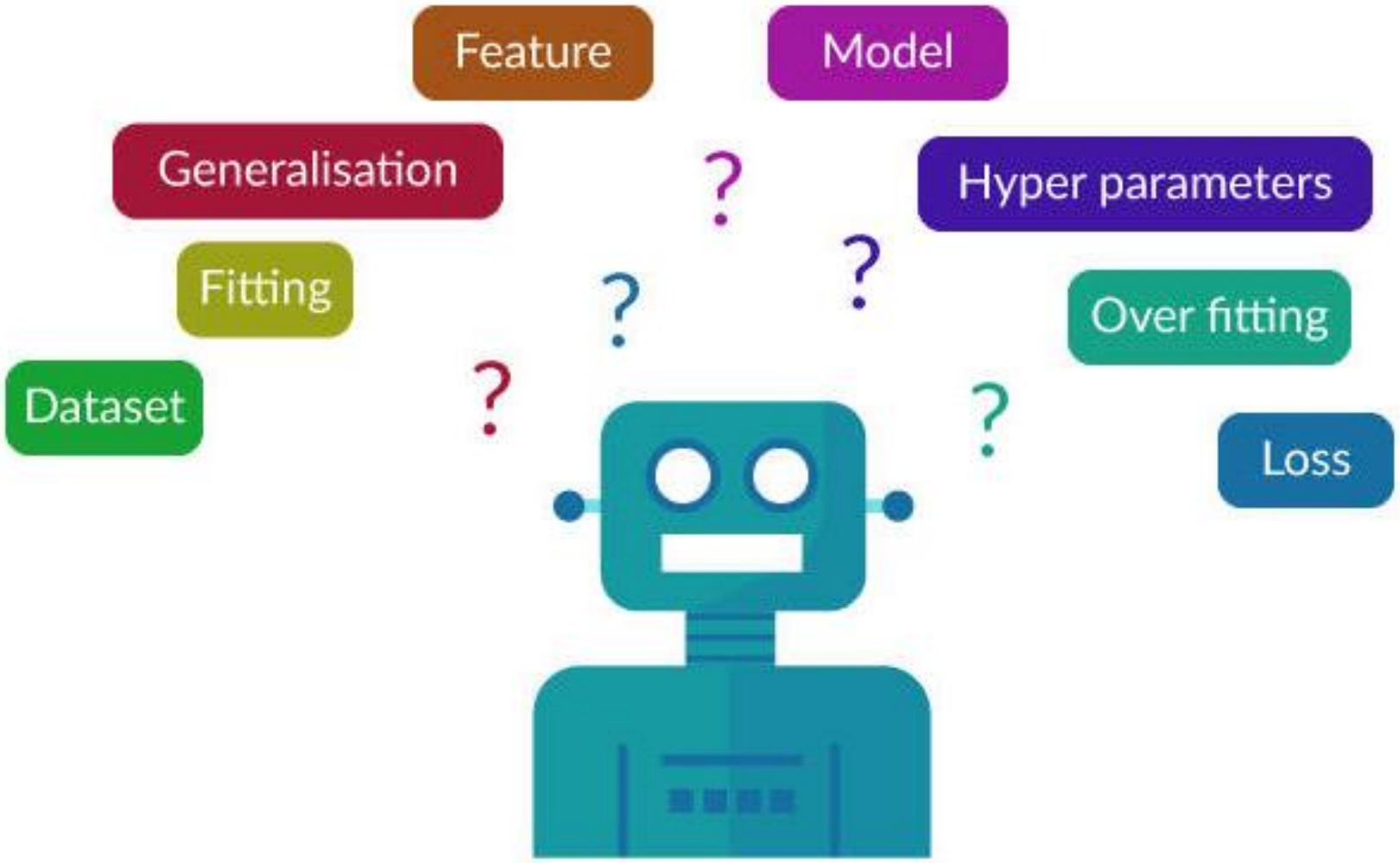
Supervised vs Unsupervised vs Reinforcement

SUPERVISED LEARNING

UNSUPERVISED LEARNING

REINFORCEMENT LEARNING

Frequently Used ML Terminologies



Machine Learning techniques

- Supervised Techniques
    1. Classification
    2. Regression
- Unsupervised Techniques
    1. Clustering
    2. Dimensionality Reduction
    3. Association Rule Mining
    4. Outlier Detection

Classification

- Classification is the process of identifying a category to which a new observation belongs, based on a training set of data containing observations whose categories are already known.

- It follows a two-step process, namely:
  1. **Learning Step** - Training phase where a model is constructed.
  2. **Classification Step** - Predicting the class labels and testing the same for accuracy.

- Classification predicts the value of the categorical variables.

Regression

- Regression analysis is a statistical method that aids in examining the relationship between two or more variables of interest.
- It examines the influence of one or more independent variables on a dependent variable.
- Types of Regression:
    1. Linear Regression
        a. Single Variable Linear Regression
        b. Multiple Linear Regression
    2. Polynomial Regression

Clustering

- Clustering is the task of grouping a set of objects, such that objects in the same cluster are similar to each other when compared to the objects in the other clusters.
- Distance measure plays a significant role in clustering.
- The common distance measures used in various datasets are as follows.
- Numeric Dataset
  - Manhattan distance
  - Minkowski distance
  - Hamming distance
- Non-Numeric Dataset
  - Jaccard index
  - Cosine Similarity
  - Dice Coefficient

Dimensionality Reduction

- Dimensionality reduction refers to techniques for reducing the number of input variables in training data.
- Fewer input dimensions often mean correspondingly fewer parameters or a simpler structure in the machine learning model, referred to as degrees of freedom.
- A model with too many degrees of freedom is likely to overfit the training dataset and therefore may not perform well on new data.
- It is desirable to have simple models that generalize well, and in turn, input data with few input variables.
- **Techniques:**
  - Feature Selection
  - Dimensionality Reduction

Association Rule Mining

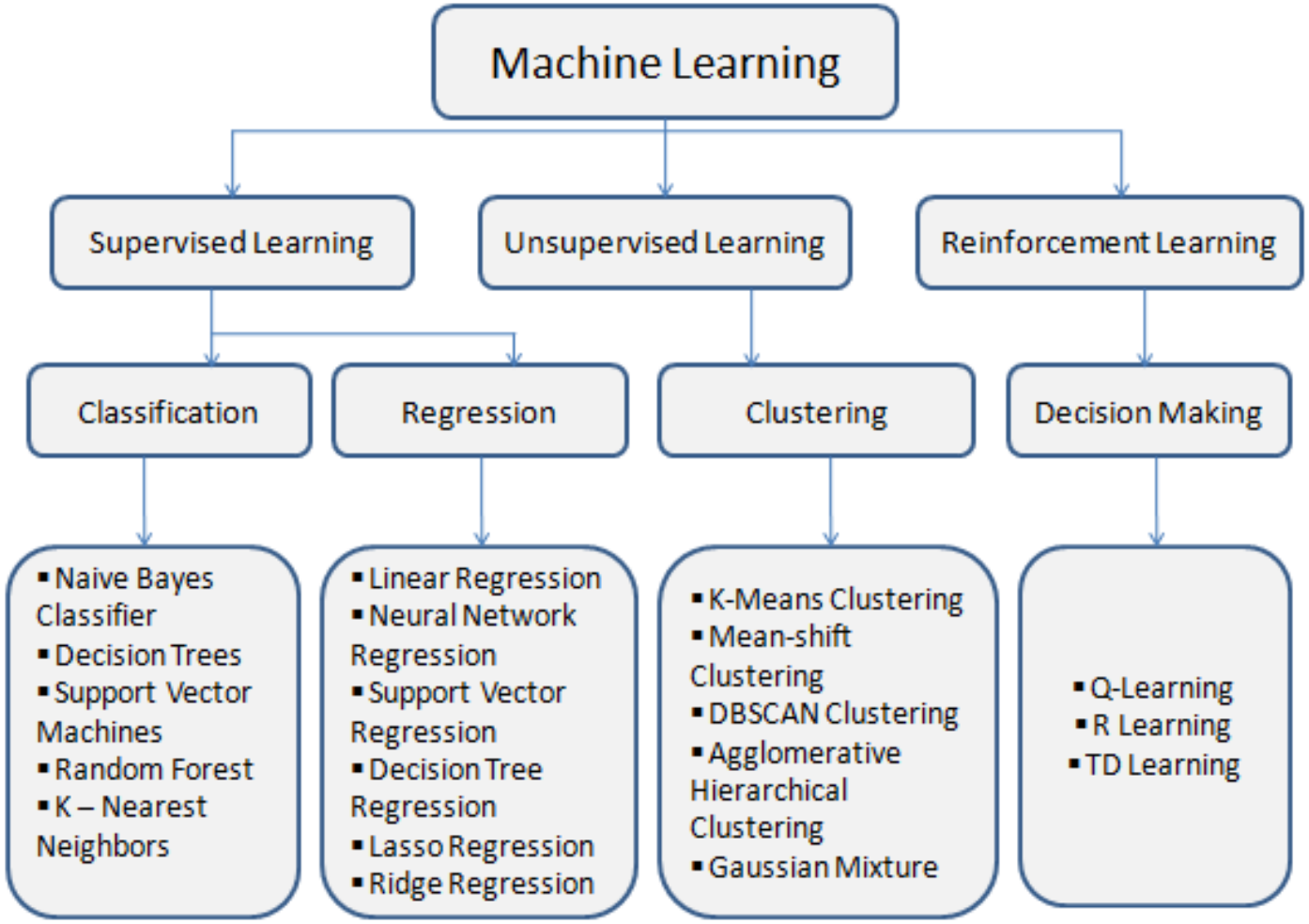
Transaction	Items purchased
1	
2	
3	
4	

Outlier Detection

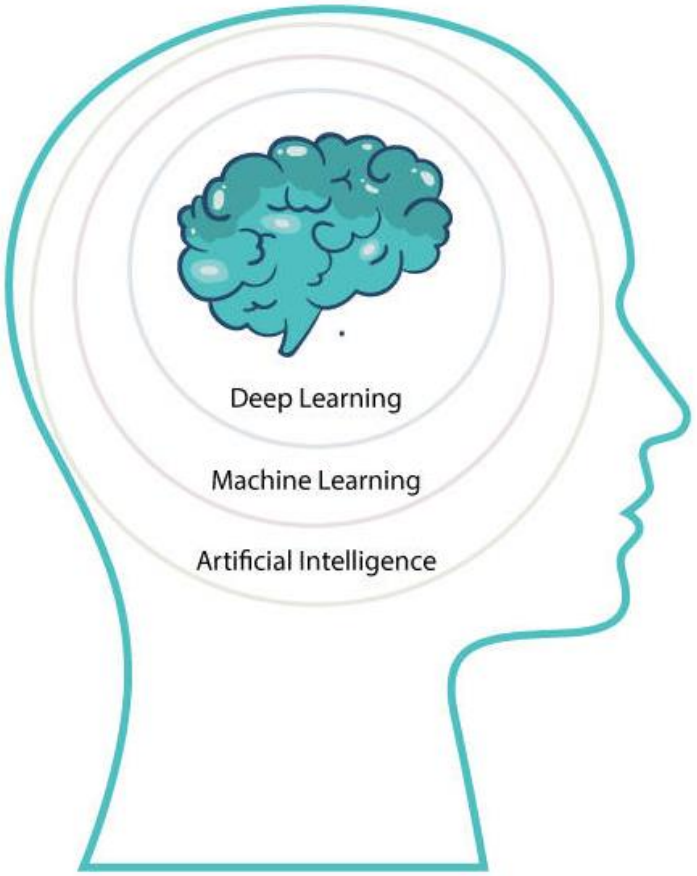
- A data object that deviates significantly from the normal objects as if it were generated by a different mechanism.

- The **types** of Outlier are as follows:
  1. **Global Outlier**: Global Outlier significantly deviates from the entire dataset.
  2. **Contextual Outlier**: Contextual Outlier significantly deviates based on the context selected.
  3. **Collective Outlier**: Collective Outlier is a subset of data objects that collectively deviates from the entire dataset.

ML Algorithms based on Technique



Deep Learning…



- Deep Learning is a type of Machine Learning that is inspired by the structure of the brain for learning **representations** of data.
- Exceptional effective at **learning patterns**.
- Deep learning algorithms attempt to learn (**multiple levels of**) representation by using a **hierarchy of multiple layers.**
- If you provide the system **tons of information**, it begins to understand it and respond in useful ways.
- Involves networks which are capable of learning from data and functions similar to the human brain.

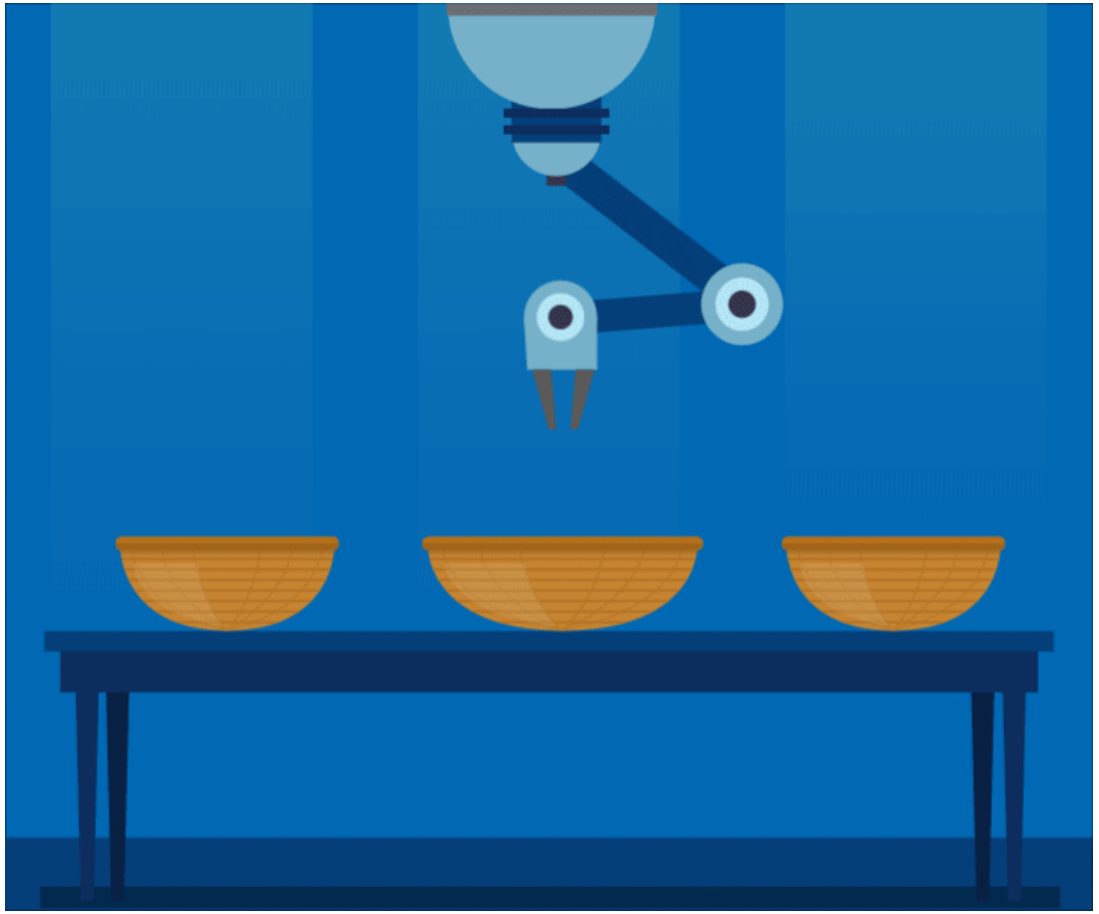
Why Deep Learning…?

- **Processes massive amount of data**
  - Deep Learning can process an enormous amount of both Structured and Unstructured data.
- **Performs Complex Operations**
  - Deep Learning algorithms are capable enough to perform complex operations when compared to the Machine Learning algorithms.
- **Achieves Best Performance**
  - As the amount of data increases, the performance of Machine Learning algorithms decreases.
  - On the other hand, Deep Learning maintains the performance of the model.
- **Feature Extraction**
  - Machine Learning algorithms extract patterns from labeled sample data, while Deep Learning algorithms take large volumes of data as input, analyze them to extract the features on its own.

Why Deep Learning…?

- Manually designed features are often **over-specified**, **incomplete** and take a **long time to design** and validate
- Learned Features are **easy to adapt**, **fast** to learn
- Deep learning provides a very **flexible**, (almost?) **universal**, learnable framework for representing world, visual and linguistic information.
- Can learn both unsupervised and supervised
- Effective **end-to-end** joint system learning
- Utilize large amounts of training data

Deep Learning vs. Machine Learning



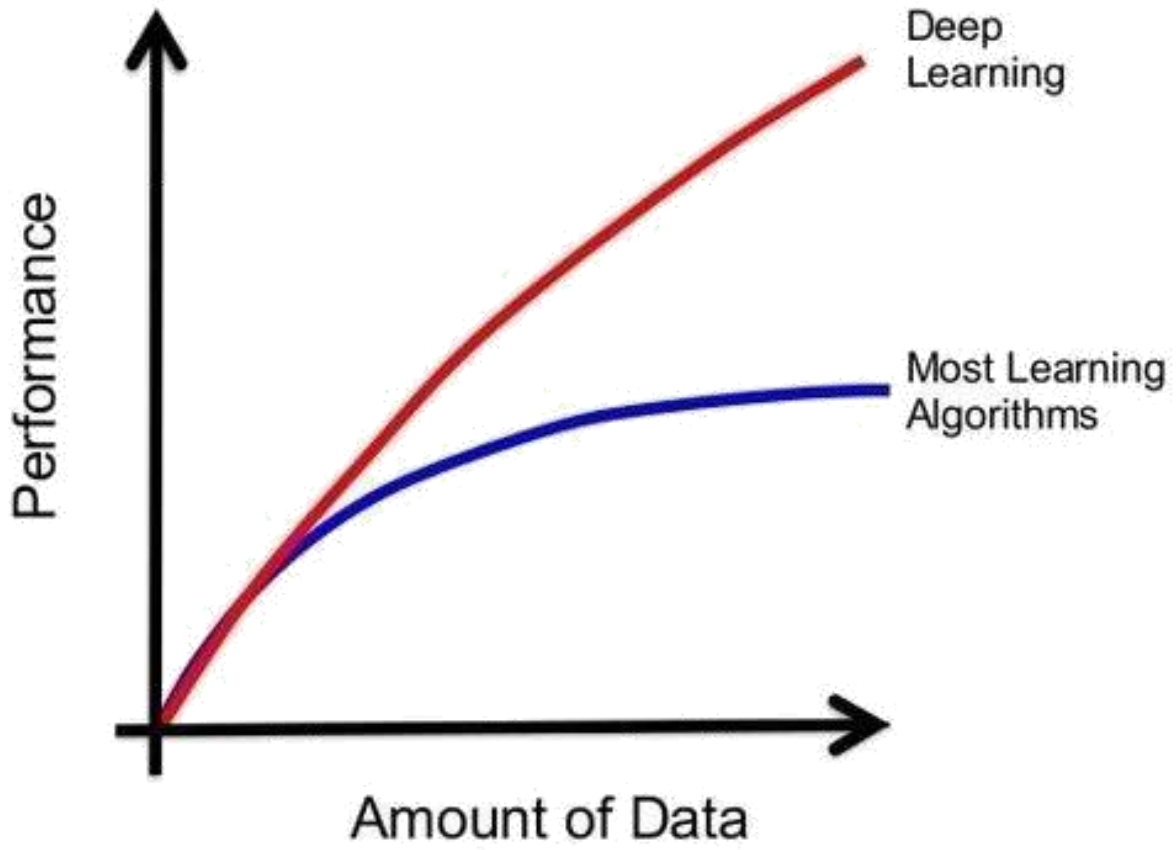
Say, for instance, we develop a machine that differentiates between *cherries and tomatoes*.

Machine Learning

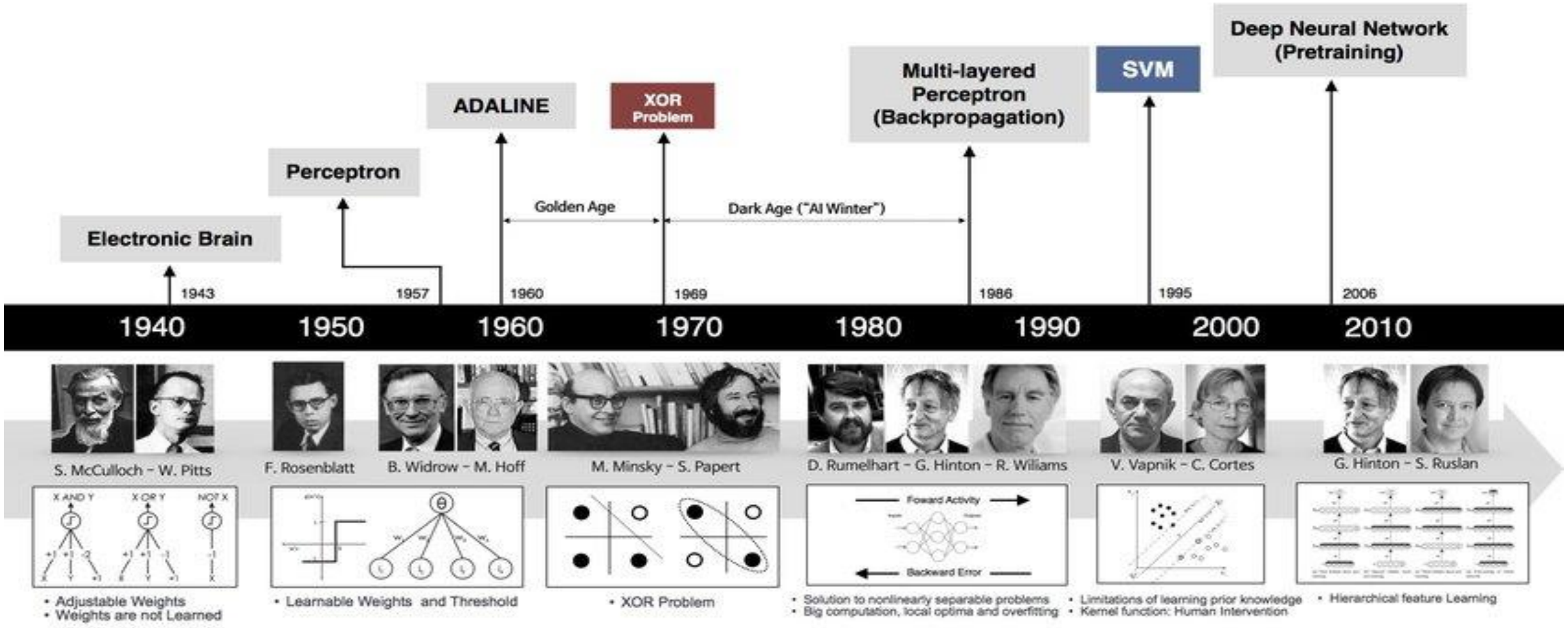
- ✓ If done through Machine Learning, ***we need to specify the features*** based on which the two can be differentiated like size and stem, in this case.

- **Deep Learning**
  - ✓ In Deep Learning, the ***features are picked by the Neural Network*** without any human intervention. But, that kind of independence can be achieved by a higher volume of data in training the machine.

Amount of Data vs Performance in Deep Learning



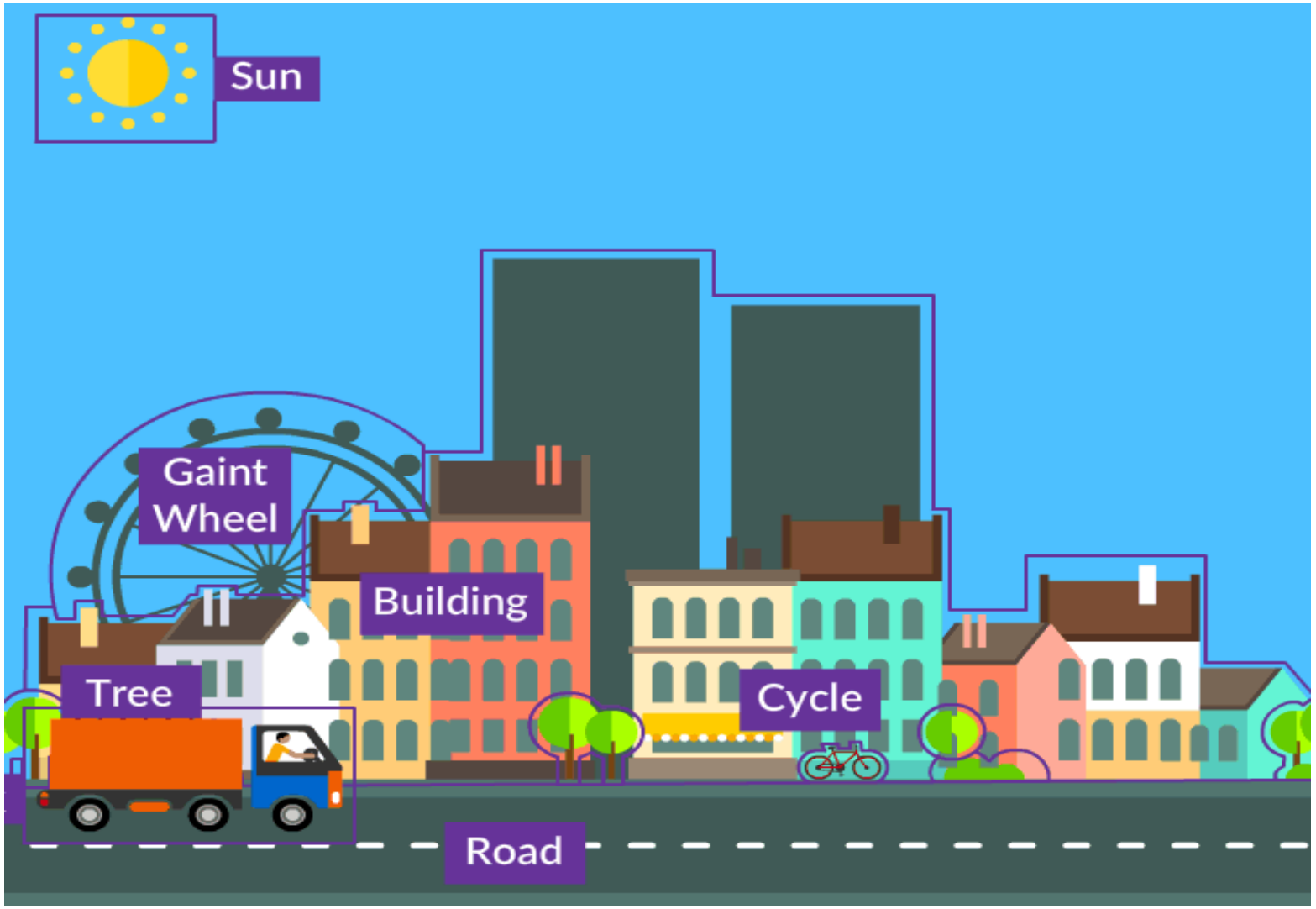
History of Deep Learning



Applications: Image Analysis

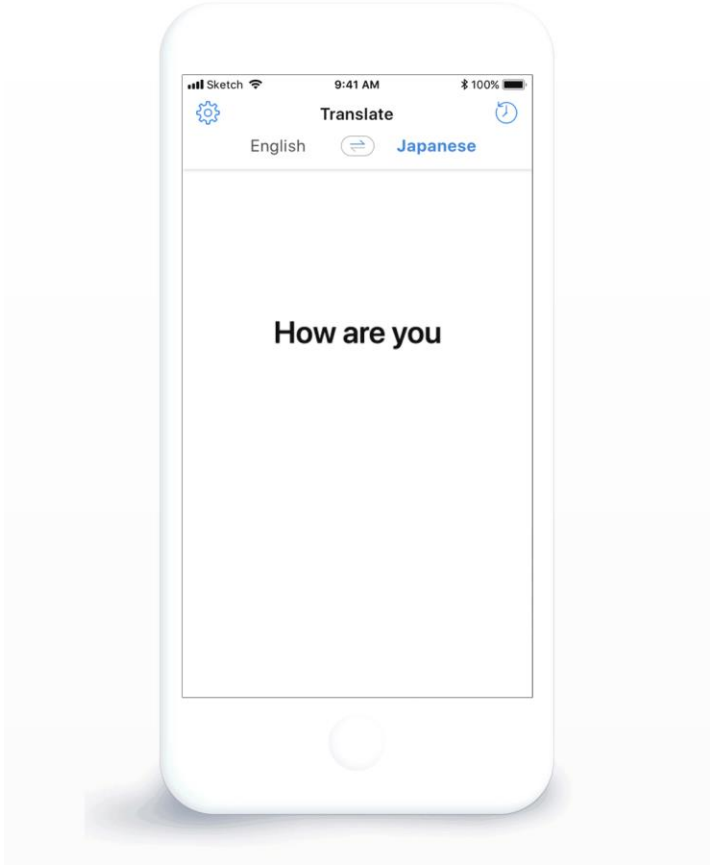
- Image Classification
- Object Recognition
- Semantic Segmentation
- Instance Segmentation

Applications: Video Analysis



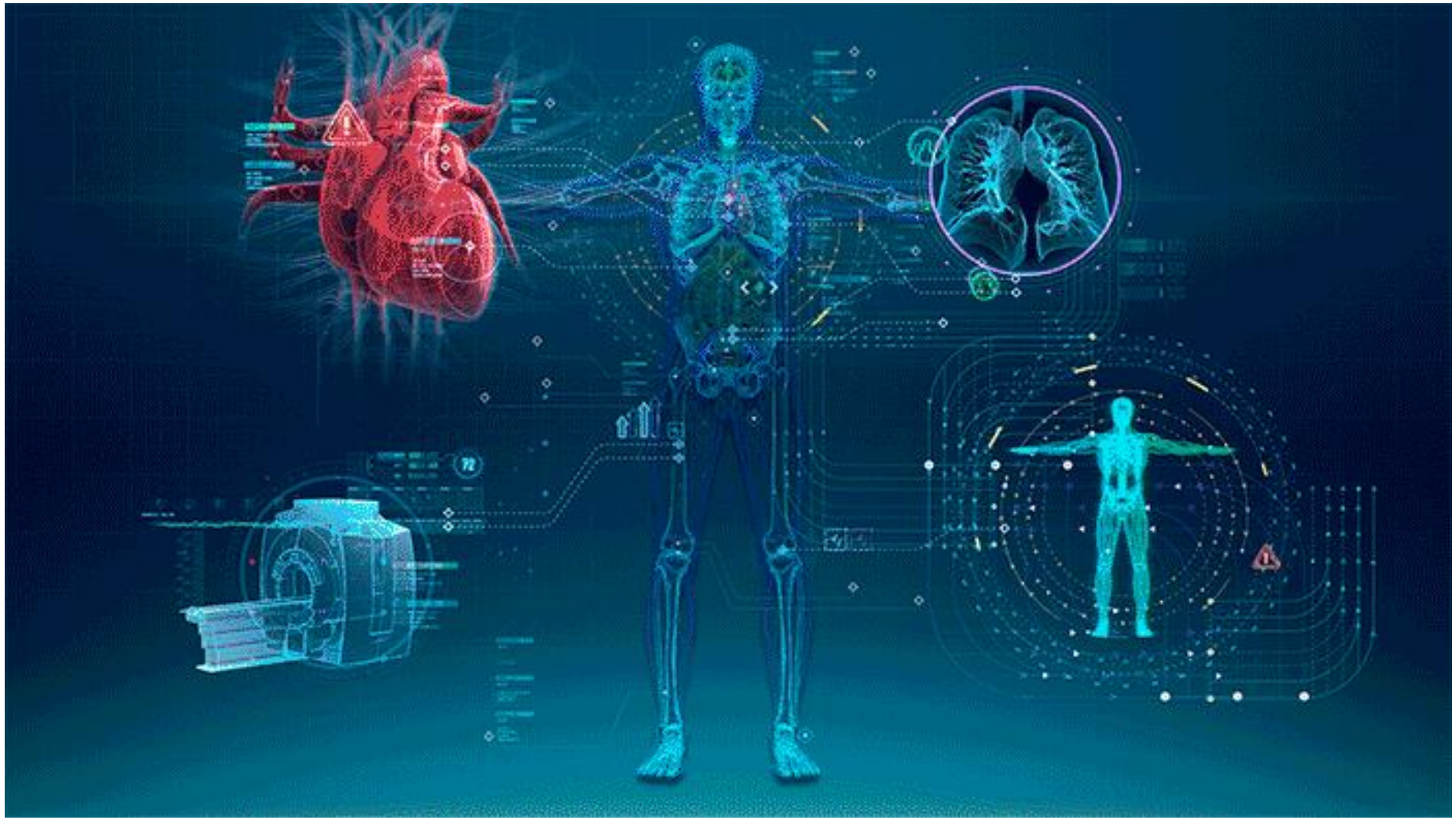
- Video Classification
- Video Recognition
- Video Captioning

Applications: Text Analysis



- Machine Translation
- Sentiment Analysis
- Document Classification
- Question Answering
- Conversational Boats

Applications: Medical Image Analysis



Deep Learning Libraries

- Libraries are a predefined set of functions and modules that can be included in our code to achieve a particular task.
- There are plenty of libraries in deep learning, each with its advantages and limitations.
- This topic covers the brief overview of deep learning libraries like Theano, Deeplearning4j, Torch, Caffe, and TensorFlow.

Theano

- Theano is a Python library.
- It was developed by a machine learning group headed by Yoshua Bengio at the University of Montreal.
- It is more a research platform than a deep learning library.
- You must perform more work by yourself to generate the models that you want.
- It does not have any deep learning classes within itself.

Theano - Features

- Theano allows us to define mathematical expressions as a set of vectors and matrices. This avoids too many for loops in our code and greatly reduces the computation time.
- Theano is best suited when we are going to build everything from scratch. It just aids in representing our deep net in terms of vectors and matrices.
- Some of the optimization techniques used by Theano are the use of GPU for computations, arithmetic simplification, constant folding, using memory aliasing to avoid calculation, etc.
- Popular libraries like Keras, Lasagne, Blocks, and Pylearn2 are built on top of Theano.

DeepLearning4J

- DeepLearning4J (DL4J) is a Deep Learning framework created in Java and JVM languages for using in commercial deep learning projects.
- Adam Gibson developed DL4J.
- DL4J is utilized in business environments on distributed CPUs and GPUs, making it ideal for commercial-grade applications.

DeepLearning4J - Features

- DL4J runs on distributed GPUs and CPUs.
- It allows us to tune the deep net by selecting values for hyper parameters
- It supports most of the deep nets like DBN, RBN, CNN, RNTN, auto encoders, Recurrent net and vanilla MLP.
- It also includes vectorization library called Canova and distributed multi-node map reduce procedure for training the model.

Torch

- Torch is a Lua deep learning framework developed by Koray Kavukcuoglu, Clement Farabet and Ronan Collobert for research and development activities into deep learning algorithms.
- Torch is written in LuaJit (framework in Lua programming language) with an underlying C implementation.
- It has also been further contributed by Facebook, Google DeepMind, Twitter and a host of others.
- Popular applications of Torch are for supervised image problems with Convolutional Neural Networks and agents in more complex domains with deep reinforcement learning.

Torch - Features

- Torch allows us to set up, train, and model deep net by configuring its hyperparameters.
- Fast and efficient GPU support.
- Provides built-in functions for indexing, transposing, slicing and numeric optimization.
- Embeddable, with ports to iOS and Android backends.

Caffe

- Caffe library was developed by Yangqing Jia at the Berkeley Vision and Learning Center for supervised computer vision problems.
- It is written in C++ with a Python interface
- It is mainly suited for machine vision tasks and also supports speech and text, reinforcement learning and recurrent nets.

Caffe - Features

- An application can easily switch between CPU and GPU since Caffe is written in C++ with CUDA (a parallel computing platform).
- You can build extremely complex deep net since it allows to perform highly sophisticated configuration on each layer.
- It also includes Matlab and Python interfaces.
- Caffe stores, communicates and manipulates the information as blobs that provides synchronization between CPU and GPU.

Tensor Flow

- Tensor Flow is an open-source deep learning library from Google.
- It is available on GitHub.
- Its flexible architecture permits to use computation to one or more GPUs or CPUs in a server, desktop or mobile device with one API.

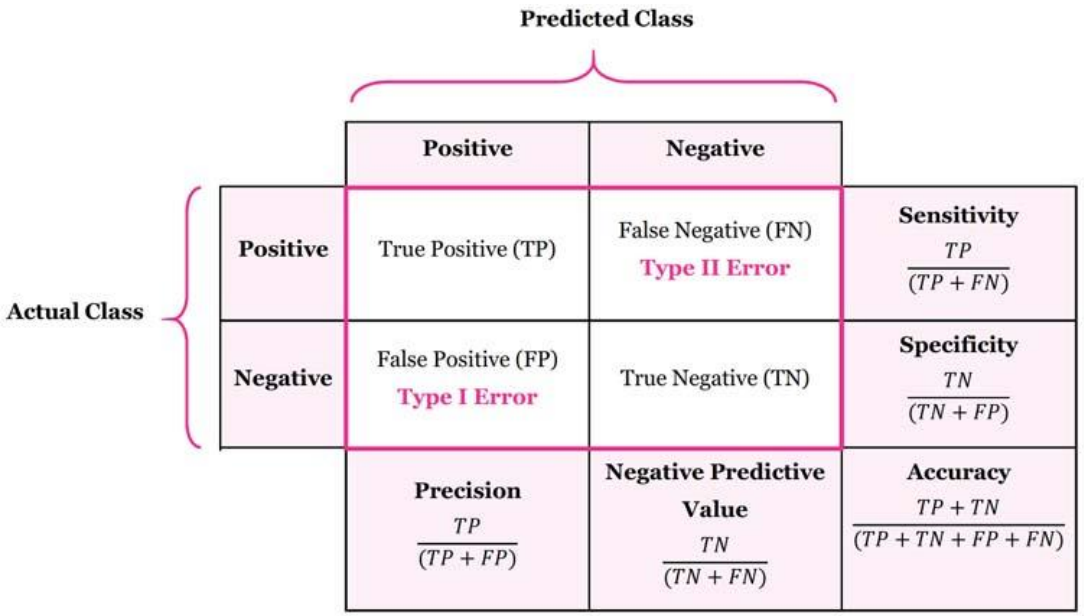
Tensor Flow - Features

- It is similar to the computational graph where edges carry data as N-dimensional vector known as tensor and nodes represent mathematical operation which acts on the tensors.
- Tensor flow lets you deploy parallel computing devices to execute operations quicker. Operations at the nodes are automatically scheduled for parallel computing.
- It includes features such as auto differentiation, shared and symbolic variables, and common subexpression elimination.
- It comes with visualization tools for graphically viewing different levels of the network, changes over time throughout the training process.

Metrics

- Metrics are used to measure the performance of a neural network or any other model.
- The commonly used metrics are error, accuracy, precision, and recall, F1-score.
- Error is the ratio of the number of incorrect classification by a total number of classifications made by the net. This metric may be misleading if data is skewed over one class over other.
- Accuracy, precision, recall and F1-score values are derived from confusion matrix (a table that is often used to describe the performance of a classification model) to know how well your model classifies the data.

Confusion Matrix



Precision:

- Out of predicted classification, what proportion actually predicted correctly?
- True positive/(True positive + False positive)

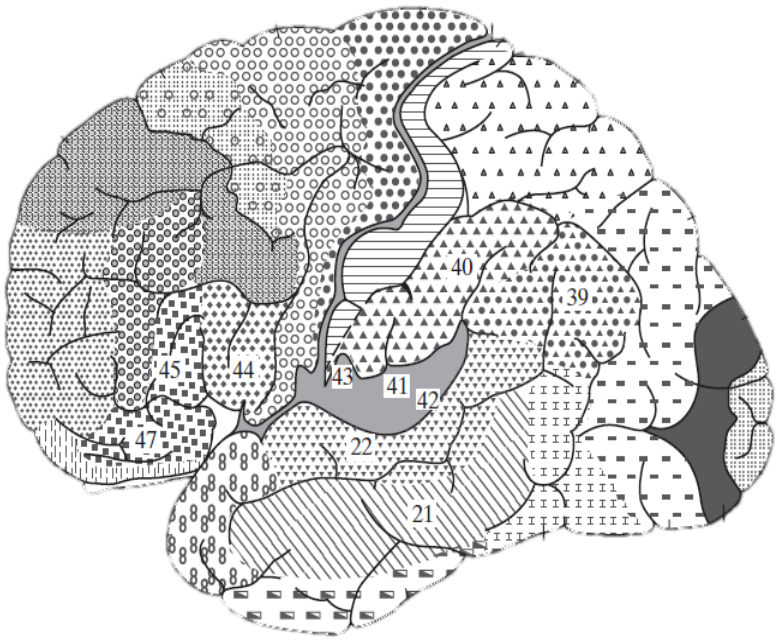
Recall:

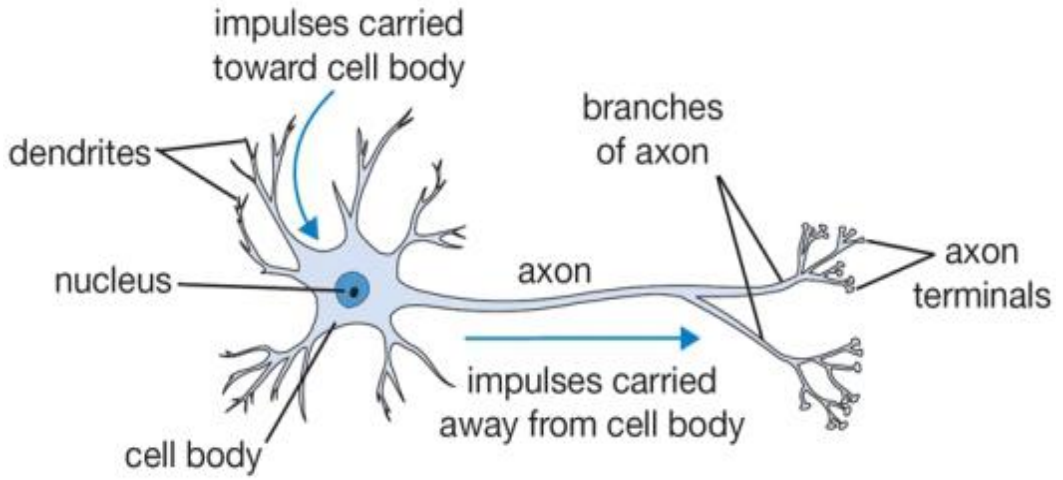
- out of an actual number of classification, what proportion were classified correctly?
- True positive/(True positive + False negative)

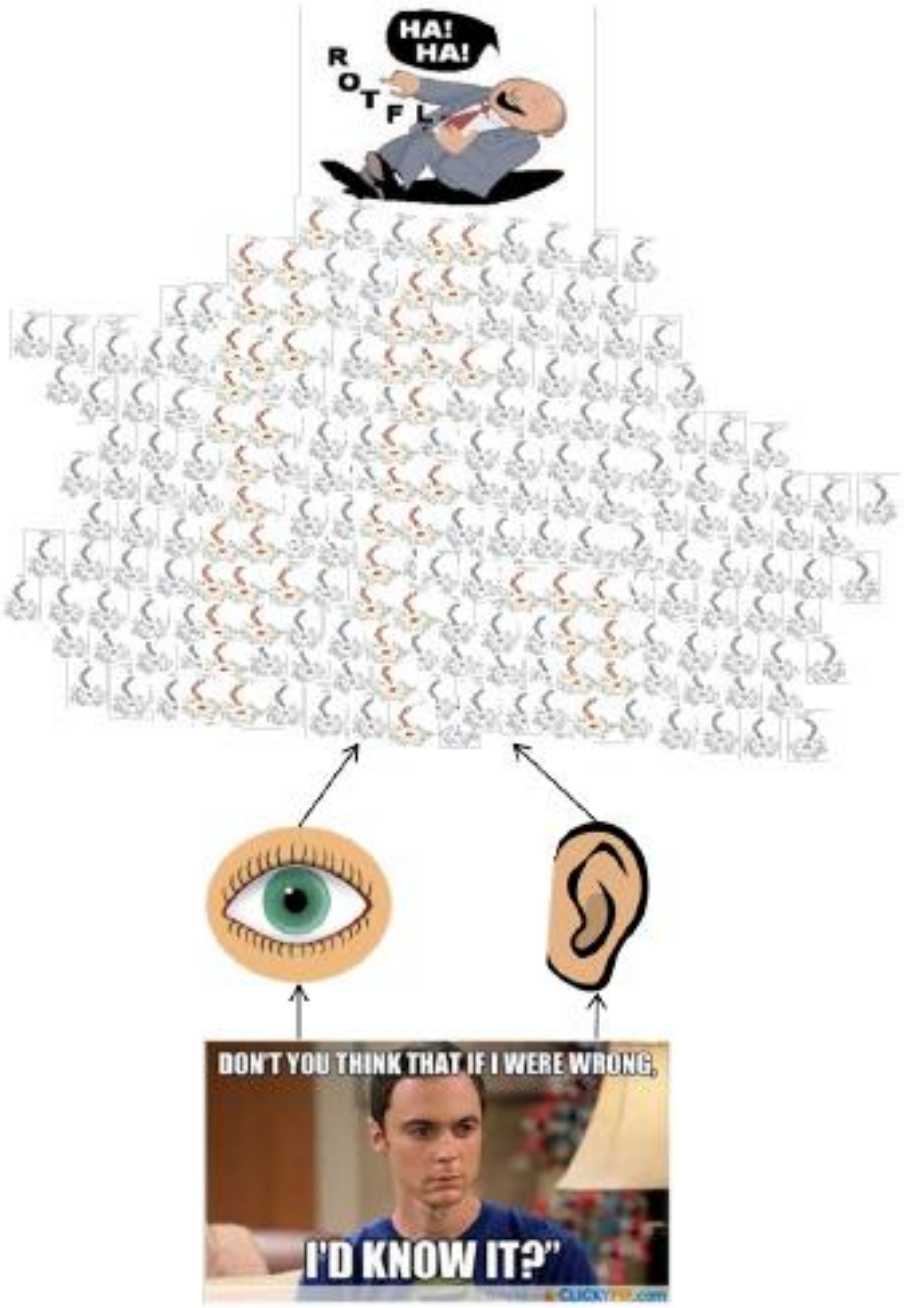
Human Information Processing System

Human Information Processing System

- The human brain contains billions of cells called Neurons. The structure of a neuron is depicted in the above image.
- An average human brain has around 10^11 (100 billion) neurons!

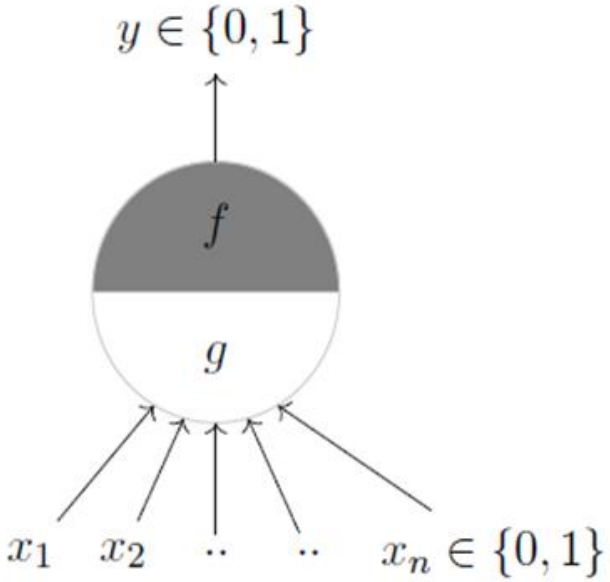






Information Flow in a Neural Network

McCulloch and Pitts (MP) Neuron



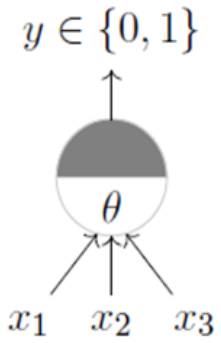
- McCulloch (neuroscientist) and Pitts (logician) proposed a highly simplified computational model of the neuron (1943)
- g aggregates the inputs and the function f takes a decision based on this aggregation
- The inputs can be excitatory or inhibitory
- $y = 0$ if any x_i is inhibitory, else

$$g(x_1, x_2, ..., x_n) = g(\mathbf{x}) = \sum_{i=1}^{n} x_i$$

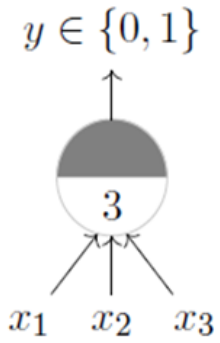
$$y = f(g(\mathbf{x})) = 1 \quad if \quad g(\mathbf{x}) \geq \theta$$
$$= 0 \quad if \quad g(\mathbf{x}) < \theta$$

- θ is called the thresholding parameter
- This is called Thresholding Logic

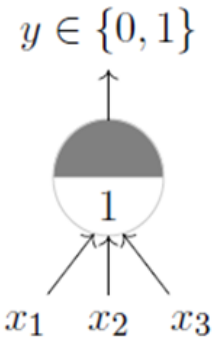
Realizing Boolean Logics using MP Neuron



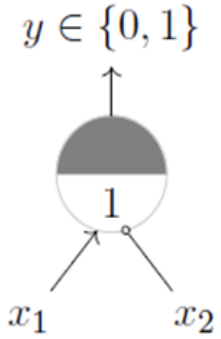
A McCulloch Pitts unit



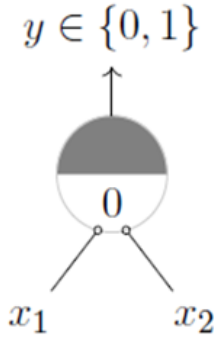
AND function



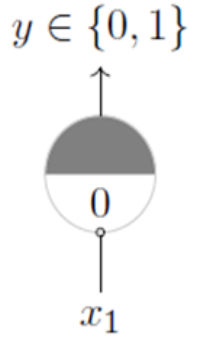
OR function



x_1 AND $!x_2$*



NOR function



NOT function

Implementing MP Neuron in Python

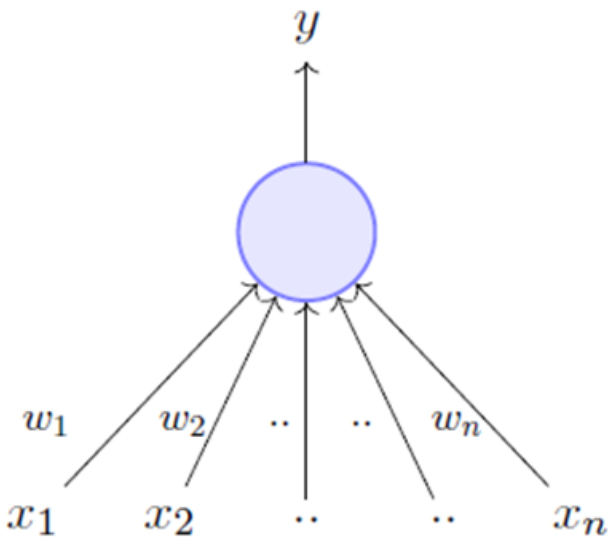
Conclusion

A single McCulloch Pitts Neuron can be used to represent Boolean functions which are linearly separable

Problems with MP Neuron...

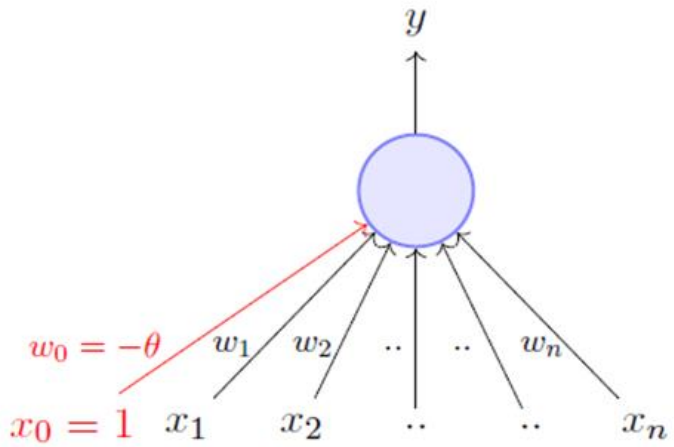
1. What about non Boolean (say, real) inputs ?
2. Do we always need to hand code the threshold ?
3. Are all inputs equal ? What if we want to assign more weight (importance) to some inputs ?
4. What about functions which are not linearly separable ?

Perceptron



- Frank Rosenblatt, an American psychologist, proposed the **classical perceptron** model (1958)
- A more general computational model than McCulloch–Pitts neurons
- **Main differences:** Introduction of numerical weights for inputs and a mechanism for learning these weights
- Inputs are no longer limited to boolean values
- Refined and carefully analyzed by Minsky and Papert (1969) - their model is referred to as the **perceptron** model here

Perceptron



A more accepted convention,

$$y = 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq 0$$

$$= 0 \quad if \sum_{i=0}^{n} w_i * x_i < 0$$

$where, \quad x_0 = 1 \quad and \quad w_0 = -\theta$

$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i \geq \theta$$

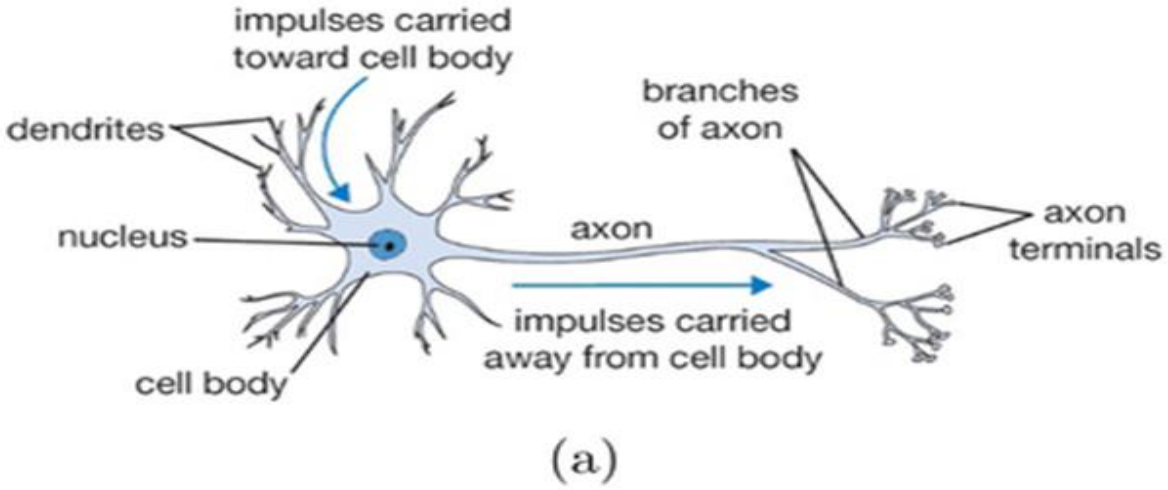
$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i < \theta$$

Rewriting the above,

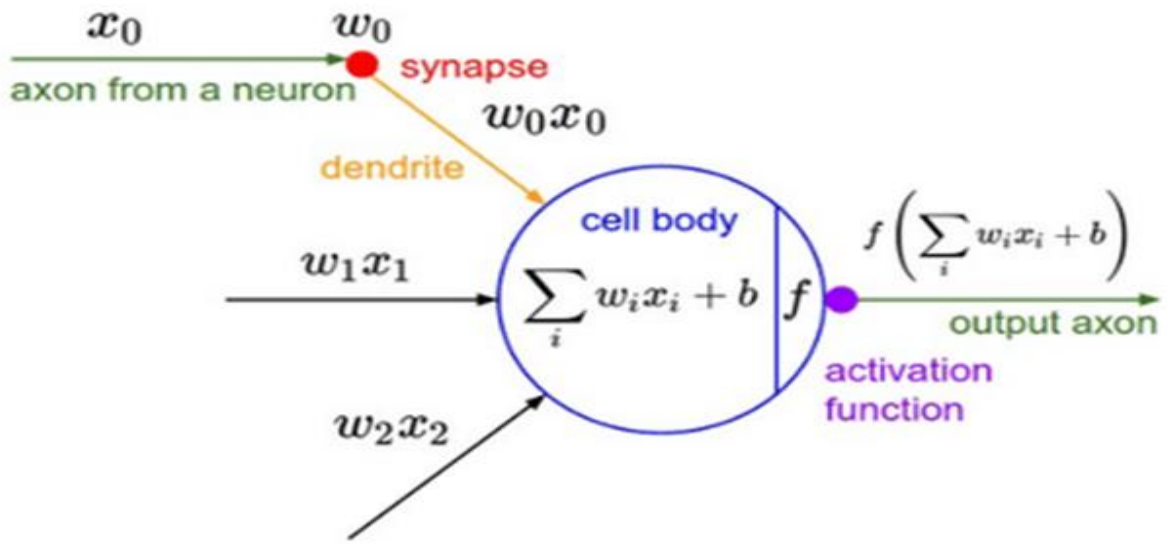
$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i - \theta \geq 0$$

$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i - \theta < 0$$

Biological vs Artificial Neurons



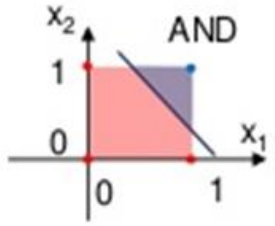
(a)

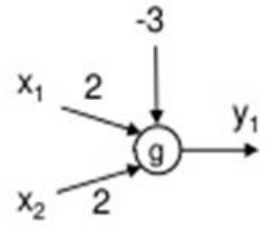


(b)

AND, OR Logic with Perceptron

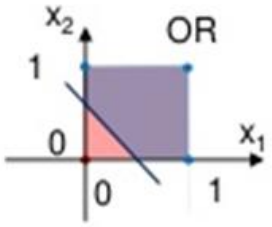
Input vector (x1,x2)	Class AND
(0,0)	0
(0,1)	0
(1,0)	0
(1,1)	1

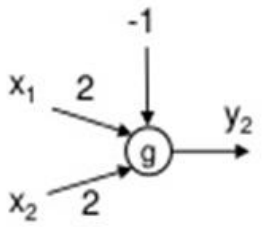




$$y_1 = g(\mathbf{w}^T\mathbf{x} + b) = u((2 \quad 2) \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - 3)$$

Input vector (x1,x2)	Class OR
(0,0)	0
(0,1)	1
(1,0)	1
(1,1)	1





$$y_2 = g(\mathbf{w}^T\mathbf{x} + b) = u((2 \quad 2) \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - 1)$$

Perceptron Learning Algorithm

Algorithm: Perceptron Learning Algorithm

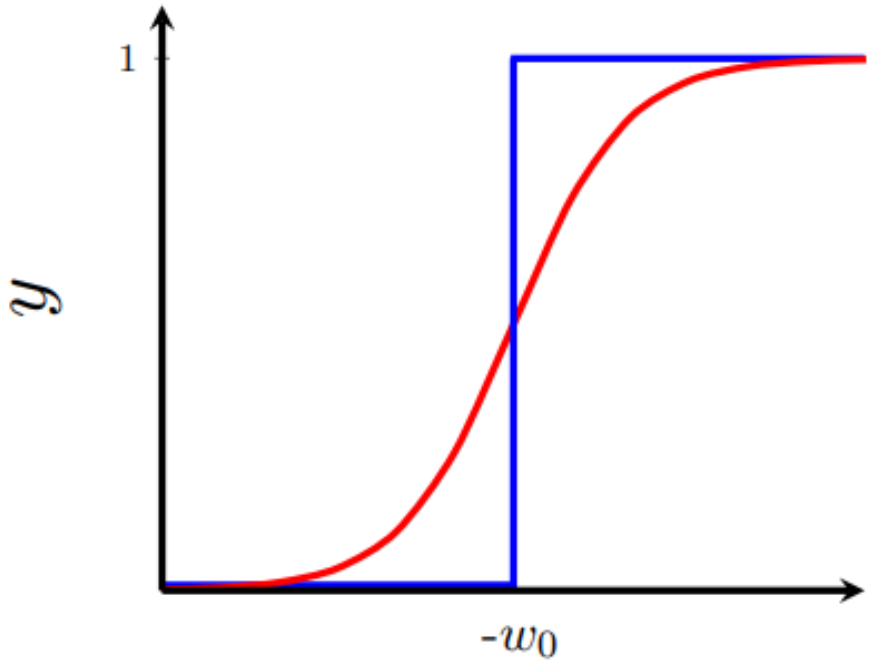
$P \leftarrow inputs \quad with \quad label \quad 1;$
$N \leftarrow inputs \quad with \quad label \quad 0;$
Initialize $\mathbf{w}$ randomly;
while $!convergence$ **do**
    Pick random $\mathbf{x} \in P \cup N$;
    if $\mathbf{x} \in P \quad and \quad \mathbf{w}.\mathbf{x} < 0$ **then**
        $\mathbf{w} = \mathbf{w} + \mathbf{x}$;
    end
    if $\mathbf{x} \in N \quad and \quad \mathbf{w}.\mathbf{x} \geq 0$ **then**
        $\mathbf{w} = \mathbf{w} - \mathbf{x}$;
    end
end
//the algorithm converges when all the inputs are classified correctly

Implementing Perceptron in Python

Intuition to Sigmoid Neuron

- Enough about boolean functions!
- What about arbitrary functions of the form $y = f(x)$ where $x \in \mathbb{R}^n$ (instead of $\{0, 1\}^n$) and $y \in \mathbb{R}$ (instead of $\{0, 1\}$) ?
- Can we have a network which can (approximately) represent such functions ?
- Before answering the above question we will have to first graduate from ***perceptrons*** to ***sigmoidal neurons*** ...

Sigmoid Neuron



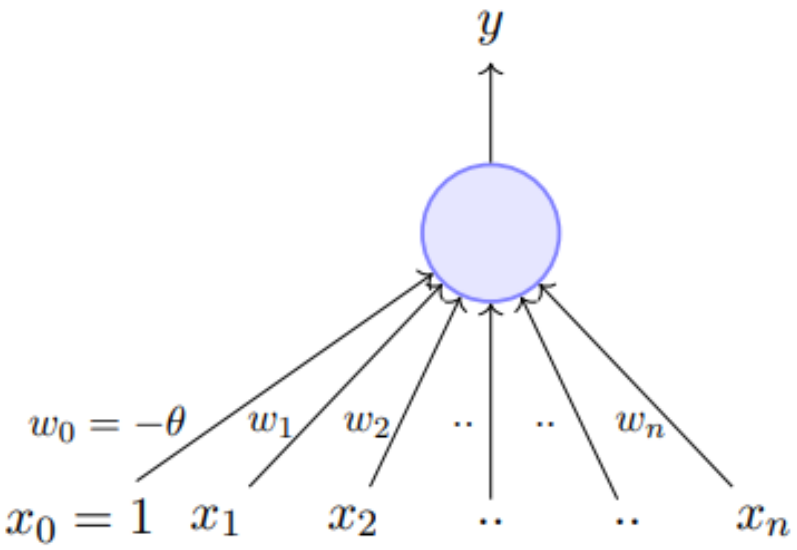
- Introducing sigmoid neurons where the output function is much smoother than the step function
- Here is one form of the sigmoid function called the logistic function

$$y = \frac{1}{1 + e^{-(w_0 + \sum_{i=1}^{n} w_i x_i)}}$$

- We no longer see a sharp transition around the threshold -w_0
- Also the output y is no longer binary but a real value between 0 and 1 which can be interpreted as a probability
- Instead of a like/dislike decision we get the probability of liking the movie

Perceptron vs Sigmoid Neuron

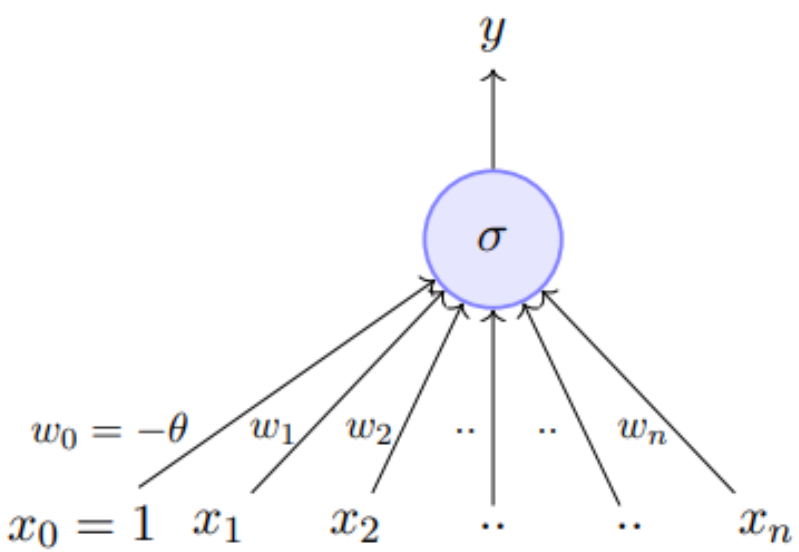
Perceptron



$$y = 1 \quad if \sum_{i=0}^{n} w_i * x_i \geq 0$$

$$= 0 \quad if \sum_{i=0}^{n} w_i * x_i < 0$$

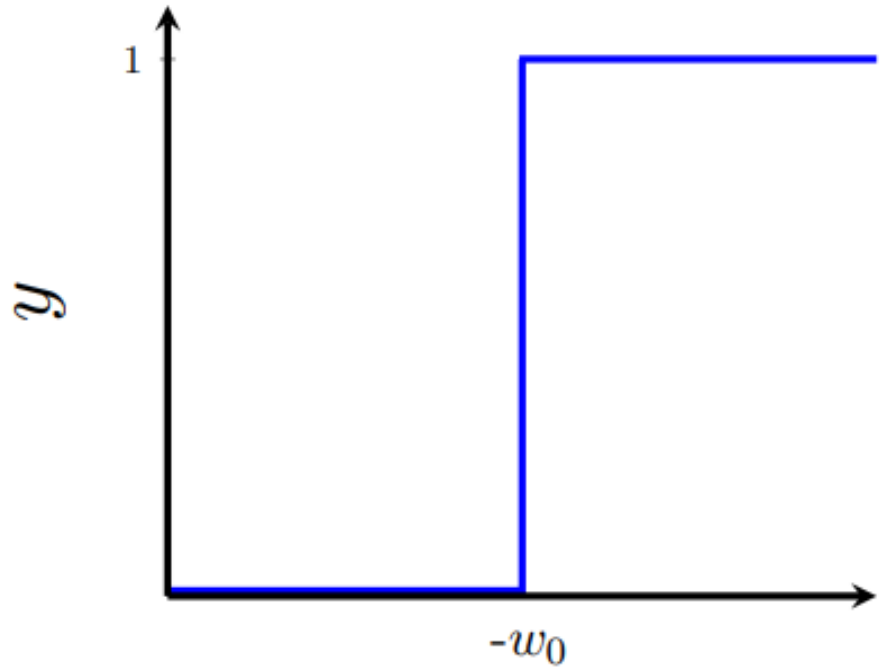
Sigmoid (logistic) Neuron



$$y = \frac{1}{1 + e^{-(\sum_{i=0}^{n} w_i x_i)}}$$

Perceptron vs Sigmoid Neuron

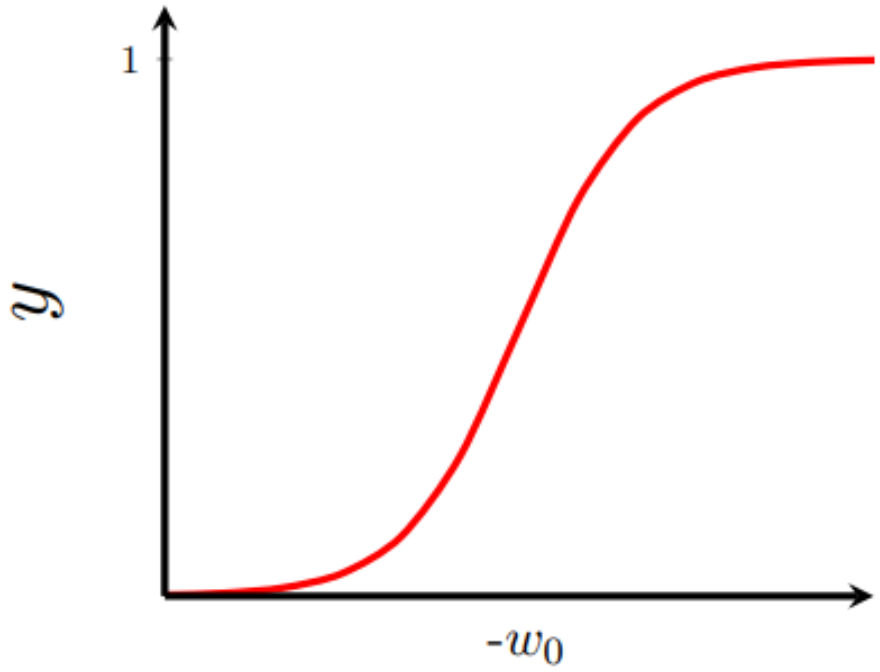
Perceptron



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

Not smooth, not continuous (at $w0$), **not differentiable**

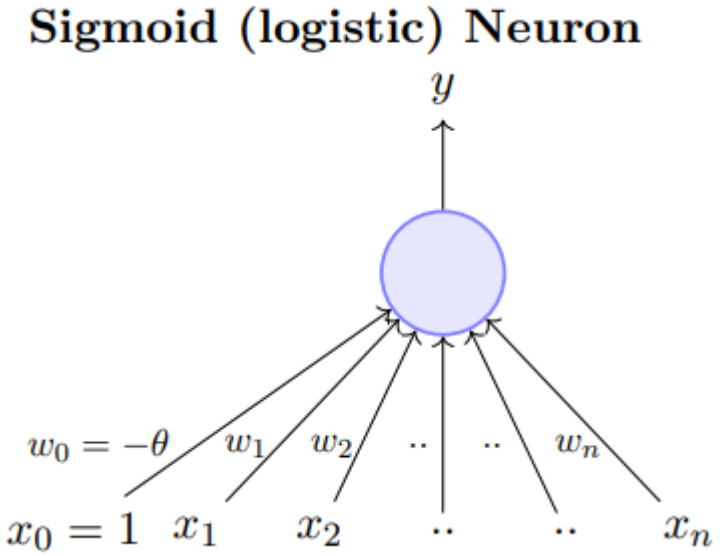
Sigmoid Neuron



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

Smooth, continuous, **differentiable**

Learning Algorithm for Sigmoid Neuron



- Well, just as we had an algorithm for learning the weights of a perceptron, we also need a way of learning the weights of a sigmoid neuron

Learning Setup

This brings us to a typical machine learning setup which has the following components...

- **Data:** $\{x_i, y_i\}_{i=1}^n$
- **Model:** Our approximation of the relation between $\mathbf{x}$ and y. For example,

$$\hat{y} = \frac{1}{1 + e^{-(\mathbf{w^T x})}}$$

$$or \quad \hat{y} = \mathbf{w^T x}$$

$$or \quad \hat{y} = \mathbf{x^T W x}$$

  or just about any function

- **Parameters:** In all the above cases, w is a parameter which needs to be learned from the data
- **Learning algorithm:** An algorithm for learning the parameters (w) of the model (for example, perceptron learning algorithm, gradient descent, etc.)
- **Objective/Loss/Error function:** To guide the learning algorithm - the learning algorithm should aim to minimize the loss function

Gradient Descent for Learning Sigmoid Neuron Params

Gradient Descent Rule

- The direction u that we intend to move in should be at 180° w.r.t. the gradient
- In other words, move in a direction opposite to the gradient

Parameter Update Equations

$$w_{t+1} = w_t - \eta \nabla w_t$$

$$b_{t+1} = b_t - \eta \nabla b_t$$

$$where, \nabla w_t = \frac{\partial \mathscr{L}(w,b)}{\partial w}_{at\ w=w_t, b=b_t}, \nabla b = \frac{\partial \mathscr{L}(w,b)}{\partial b}_{at\ w=w_t, b=b_t}$$

Gradient Descent Algorithm

Algorithm: gradient_descent()

$t \leftarrow 0$;
$max\_iterations \leftarrow 1000$;
while $t < max\_iterations$ **do**
  $w_{t+1} \leftarrow w_t - \eta \nabla w_t$;
  $b_{t+1} \leftarrow b_t - \eta \nabla b_t$;
  $t \leftarrow t + 1$;
end

$$
\begin{aligned}
\nabla w &= \frac{\partial}{\partial w}[\frac{1}{2} * (f(x) - y)^2] \\
&= \frac{1}{2} * [2 * (f(x) - y) * \frac{\partial}{\partial w}(f(x) - y)] \\
&= (f(x) - y) * \frac{\partial}{\partial w}(f(x)) \\
&= (f(x) - y) * \frac{\partial}{\partial w}\left(\frac{1}{1 + e^{-(wx+b)}}\right) \\
&= (f(x) - y) * f(x) * (1 - f(x)) * x
\end{aligned}
$$

$$
\begin{aligned}
&\frac{\partial}{\partial w}\left(\frac{1}{1 + e^{-(wx+b)}}\right) \\
&= \frac{-1}{(1 + e^{-(wx+b)})^2} \frac{\partial}{\partial w}(e^{-(wx+b)})) \\
&= \frac{-1}{(1 + e^{-(wx+b)})^2} * (e^{-(wx+b)}) \frac{\partial}{\partial w}(-(wx + b))) \\
&= \frac{-1}{(1 + e^{-(wx+b)})} * \frac{e^{-(wx+b)}}{(1 + e^{-(wx+b)})} * (-x) \\
&= \frac{1}{(1 + e^{-(wx+b)})} * \frac{e^{-(wx+b)}}{(1 + e^{-(wx+b)})} * (x) \\
&= f(x) * (1 - f(x)) * x
\end{aligned}
$$

So if there is only 1 point (x, y), we have,

$$\nabla w = (f(x) - y) * f(x) * (1 - f(x)) * x$$

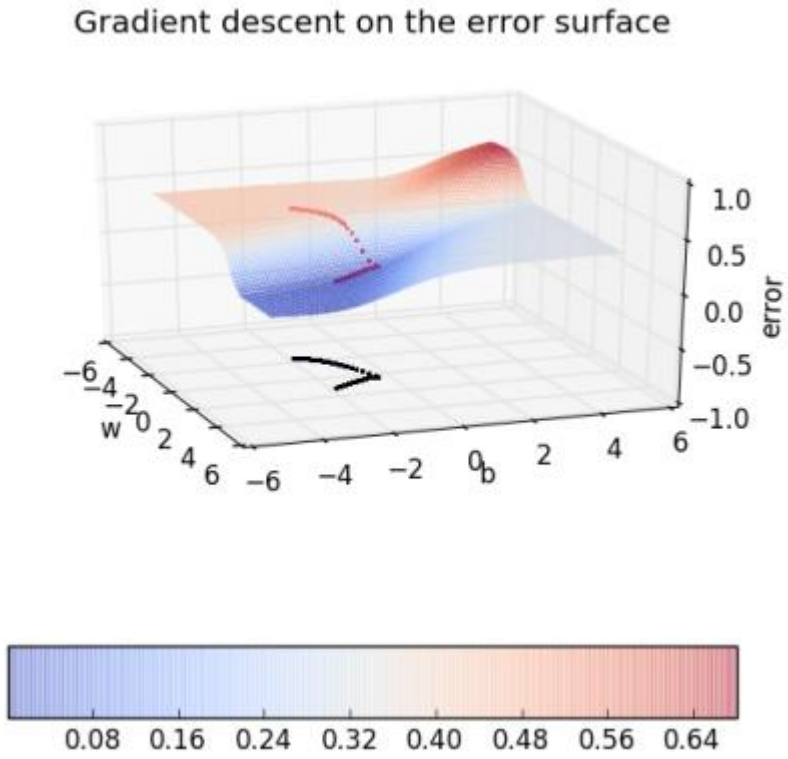
For two points,

$$\nabla w = \sum_{i=1}^{2} (f(x_i) - y_i) * f(x_i) * (1 - f(x_i)) * x_i$$

$$\nabla b = \sum_{i=1}^{2} (f(x_i) - y_i) * f(x_i) * (1 - f(x_i))$$

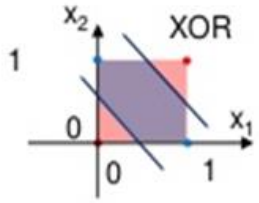
Implementing Gradient Descent

```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w,b,x) : #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x + b)))

def error (w, b) :
    err = 0.0
    for x,y in zip(X,Y) :
        fx = f(w,b,x)
        err += 0.5 * (fx - y) ** 2
    return err

def grad_b(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w,b,x,y) :
    fx = f(w,b,x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        w = w - eta * dw
        b = b - eta * db
```

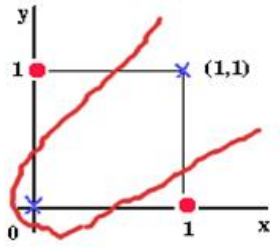


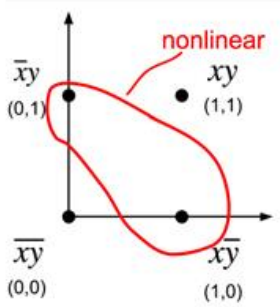
Perceptron / Sigmoid Neuron for XOR

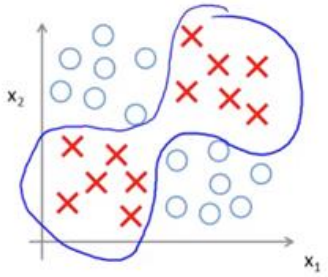
Input vector (x1,x2)	Class XOR
(0,0)	0
(0,1)	1
(1,0)	1
(1,1)	0



Find the weights of Perceptron for XOR

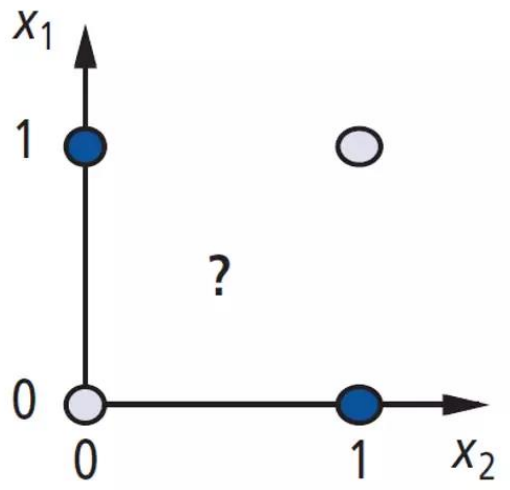


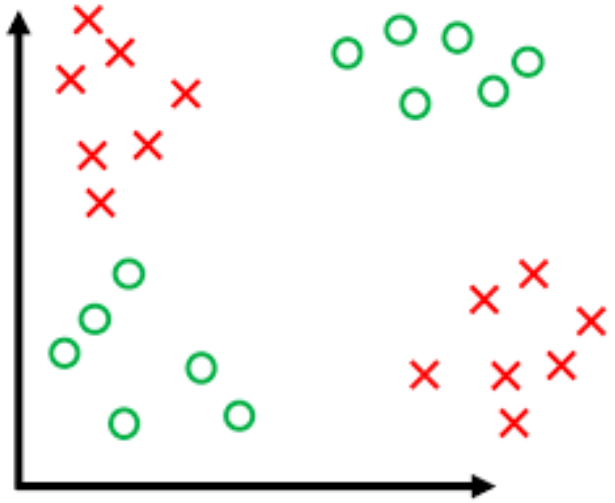




Perceptron - Failure

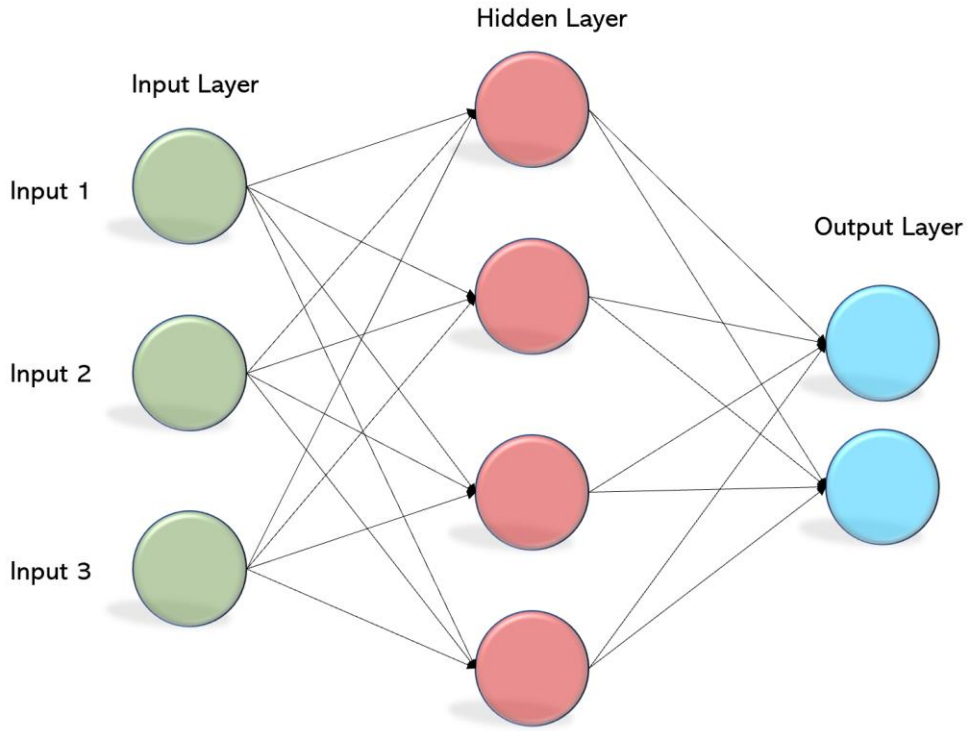
- A "single-layer" perceptron can't implement XOR.
- The reason is because the classes in XOR are not linearly separable.
- You cannot draw a straight line to separate the points (0,0),(1,1) from the points (0,1),(1,0).
- Led to invention of multi-layer networks.





Multi Layered Perceptron - MLP

- In the Multilayer perceptron, there can more than one linear layer (combinations of neurons).
- There are three different layers in a MLP / Deep Neural Network, namely:
  1. Input Layer
  2. Hidden Layer
  3. Output Layer



Structure of MLP

Input Layer:

- The input layer communicates with the external environment and presents a pattern to the neural network.
- Every neuron in the input layer represents an independent variable that influences the output.

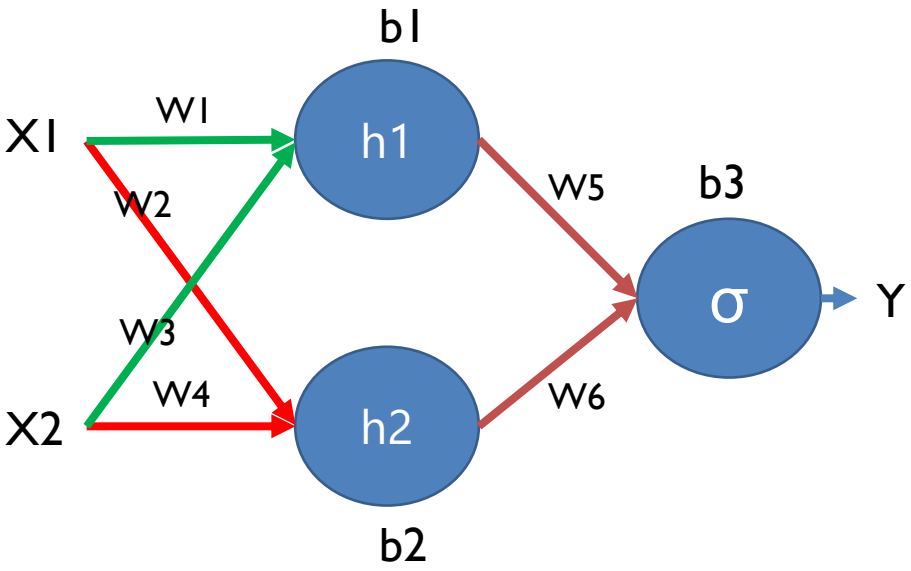
Hidden Layer:

- A Neural Network consists of several hidden layers, each consisting of a collection of neurons.
- The hidden layer is an intermediate layer found between the input layer and the output layer.
- This layer is responsible for extracting the features required from the input.
- There is no exact formula for calculating the number of the hidden layers as well as the number of neurons in each hidden layer.

Output Layer:

- The output layer of the neural network collects and transmits information in the desired format.

Learning XOR Problem



(X1, X2) = {(0,0),(1,1),(0,1),(1,0)}
Y = {0,0,1,1}
W1 = 20, W2 = -20, W3 = 20,
W4 = -20, W5 = 20, W6 = 20
b1 = -10, b2 = 30, b3 = -30

$\sigma(20x_1 + 20x_2 - 10)$

$\sigma(20*0 + 20*0 - 10) \approx 0$
$\sigma(20*1 + 20*1 - 10) \approx 1$
$\sigma(20*0 + 20*1 - 10) \approx 1$
$\sigma(20*1 + 20*0 - 10) \approx 1$

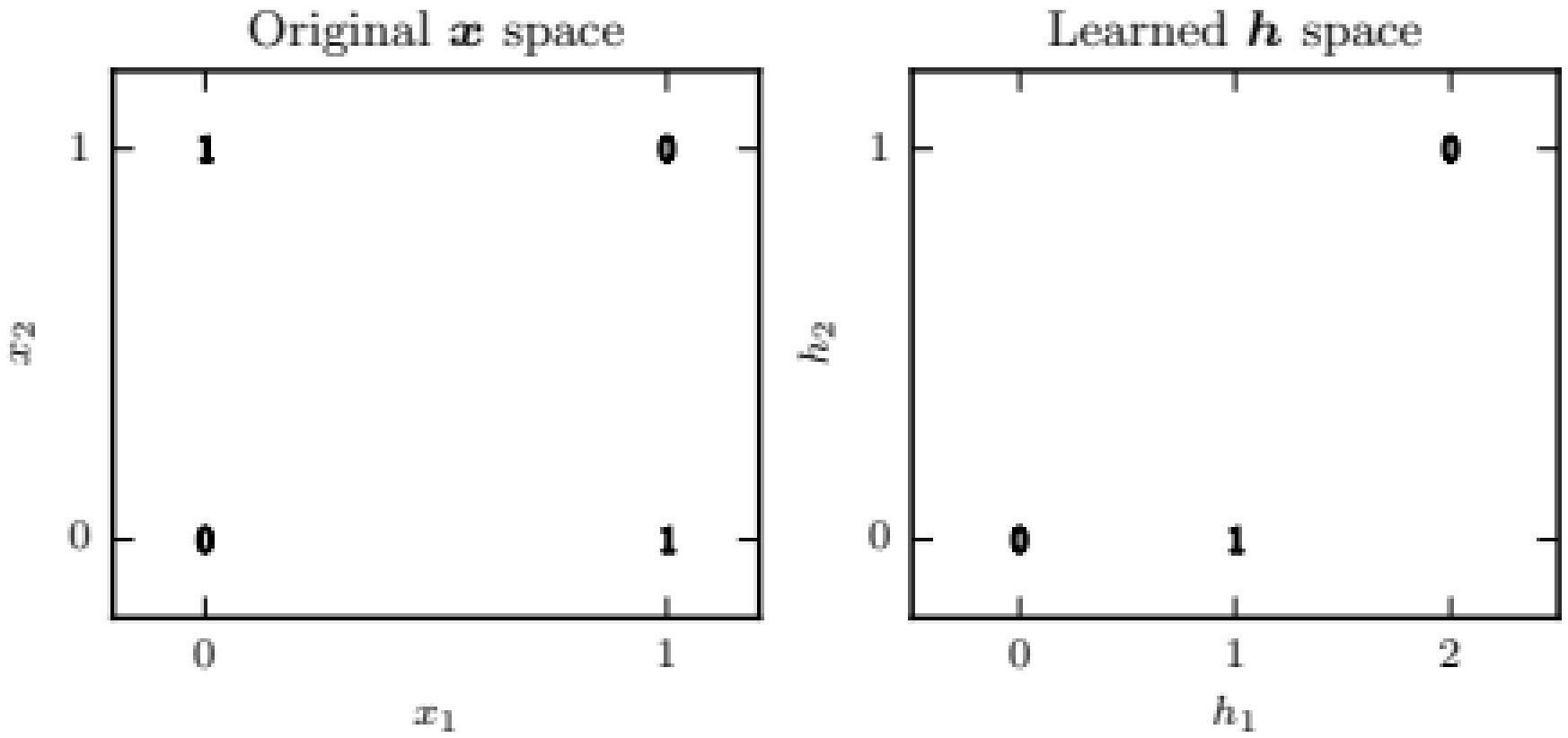
$\sigma(-20x_1 - 20x_2 + 30)$

$\sigma(-20*0 - 20*0 + 30) \approx 1$
$\sigma(-20*1 - 20*1 + 30) \approx 0$
$\sigma(-20*0 - 20*1 + 30) \approx 1$
$\sigma(-20*1 - 20*0 + 30) \approx 1$

$\sigma(20h_1 + 20h_2 - 30)$

$\sigma(20*0 + 20*1 - 30) \approx 0$
$\sigma(20*1 + 20*0 - 30) \approx 0$
$\sigma(20*1 + 20*1 - 30) \approx 1$
$\sigma(20*1 + 20*1 - 30) \approx 1$

Learning XOR Problem



Can you apply Perceptron Learning Rule for MLP…?

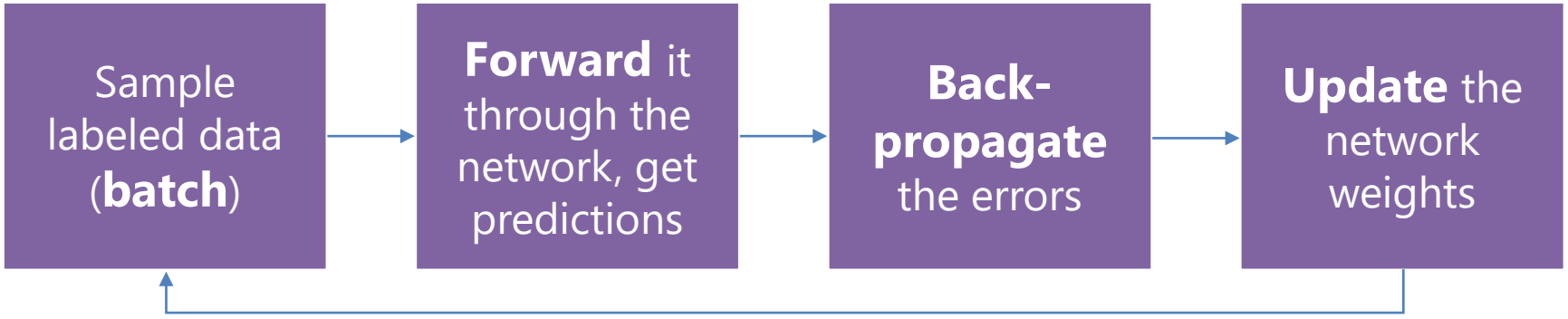
Back Propagation Learning Rule

- ***Training* of the *multilayer perceptron* proceeds in *two phases*:**
  - In the ***forward*** *phase*, the weights of the network are *fixed* and the input signal is *propagated* through the network, layer by layer, until it reaches the *output*.
  - In the ***backward*** *phase*, the error signal, which is produced by comparing the output of the network and the desired response, is *propagated* through the network, again layer by layer, but in the *backward direction*.

1. ***Initialization***
2. ***Presentation of training example***
3. ***Forward computation***
4. ***Backward computation***
5. ***Iteration***

Training Process in NN

- Optimize (min. or max.) **objective/cost function** $\boldsymbol{J(\theta)}$
- Generate **error signal** that measures difference between predictions and target values
- Use error signal to change the **weights** and get more accurate predictions
- Subtracting a fraction of the **gradient** moves you towards the **(local) minimum of the cost function**

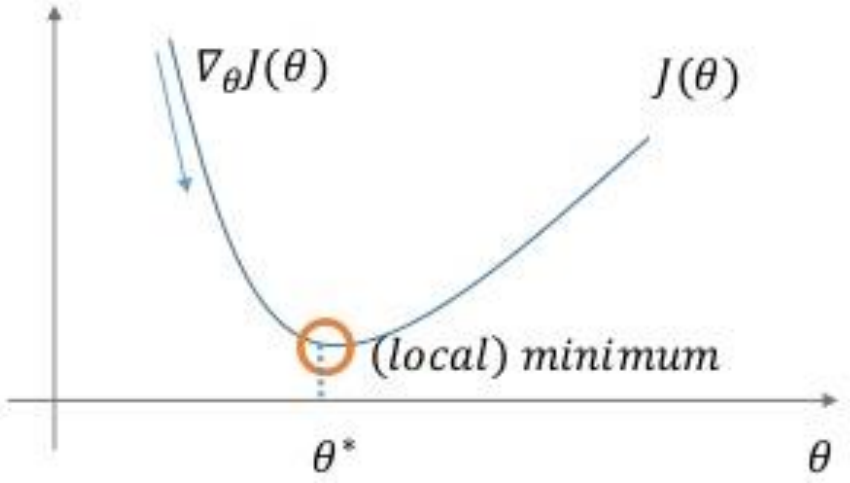


Gradient Descent – Optimization

- Gradient descent is a way to minimize an objective function $J(\theta)$
  - $J(\theta)$: Objective function
  - $\theta \in R^d$: Model's parameters
  - η : Learning rate. This determines the size of the steps we take to reach a (local) minimum.

Update equation

$$\theta = \theta - \eta * \nabla_\theta J(\theta)$$



Gradient Descent – Optimization

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

Now,

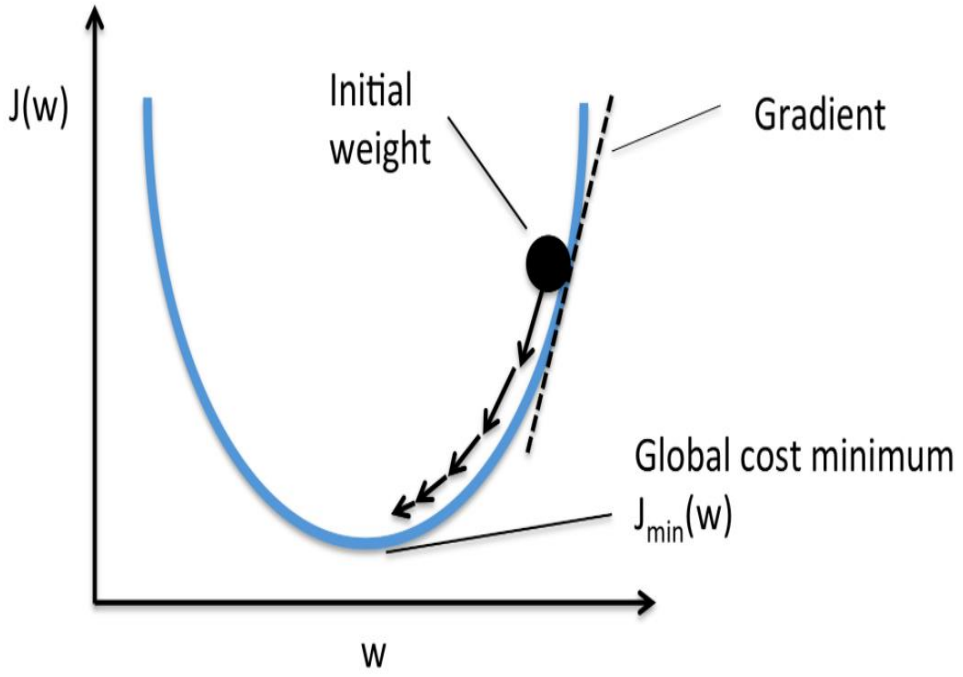
$$\frac{\partial}{\partial \theta} J_\theta = \frac{\partial}{\partial \theta} \frac{1}{2m} \sum_{i=1}^{m} \left[h_\theta(x_i) - y_i \right]^2$$

$$\frac{\partial}{\partial \theta} J_\theta = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x_i) - y_i \right) \cdot \frac{\partial}{\partial \theta_j} \left(\theta x_i - y_i \right)$$

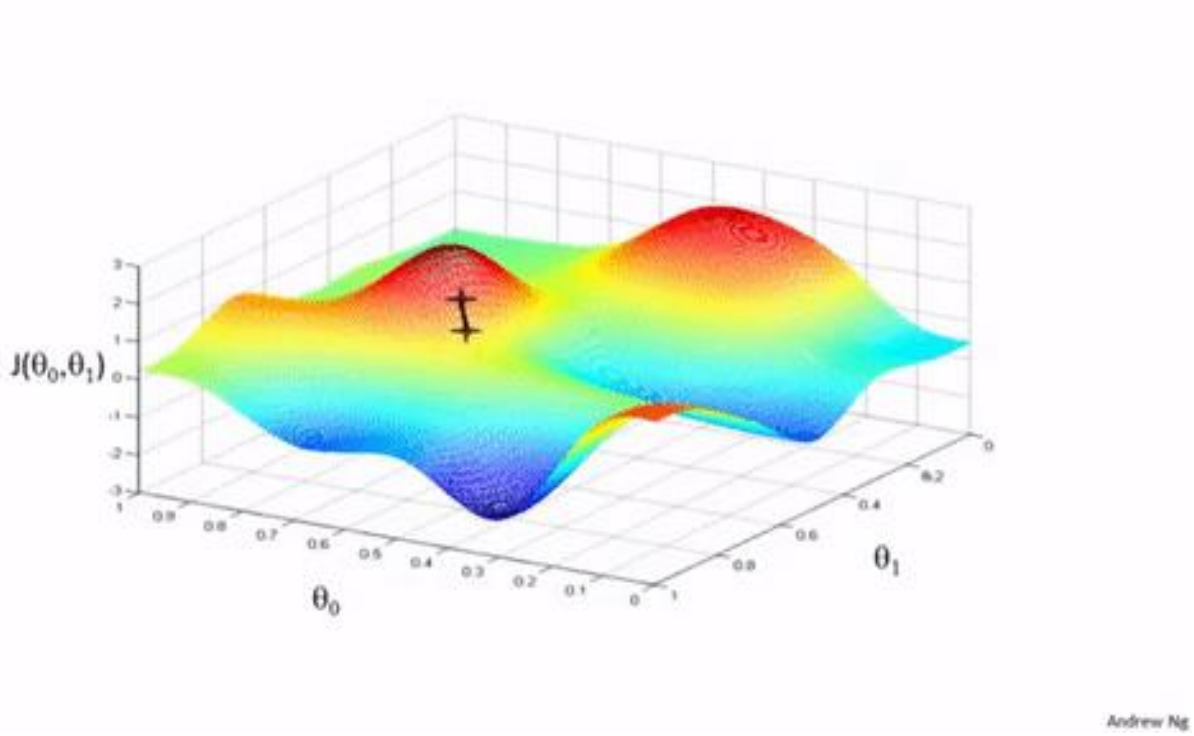
$$\frac{\partial}{\partial \theta} J_\theta = \frac{1}{m} \sum_{i=1}^{m} \left[\left(h_\theta(x_i) - y \right) x_i \right]$$

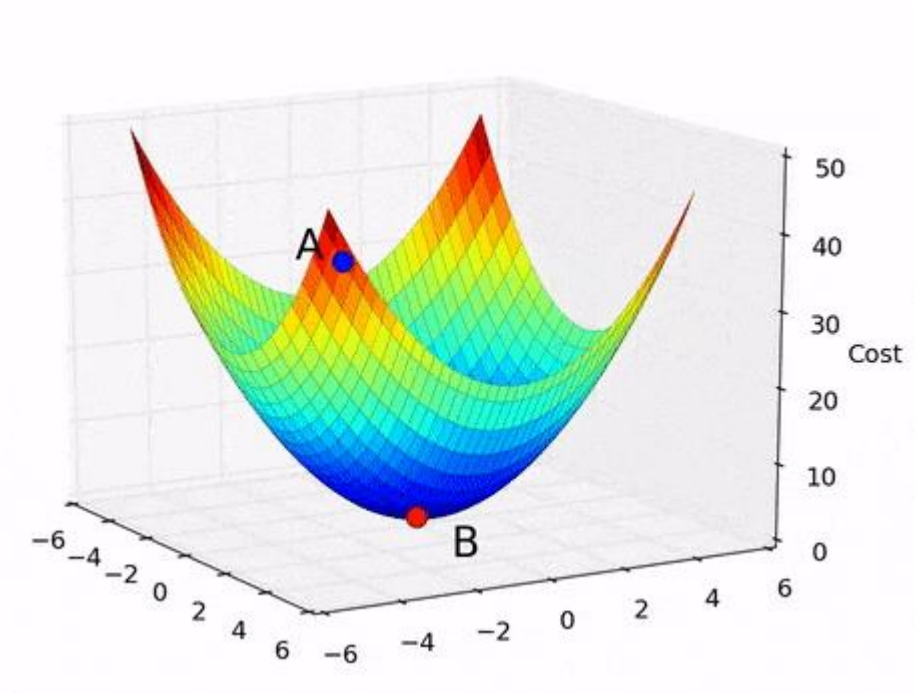
Therefore,

$$\theta_j := \theta_j - \frac{\alpha}{m} \sum_{i=1}^{m} \left[\left(h_\theta(x_i) - y_i \right) x_i \right]$$



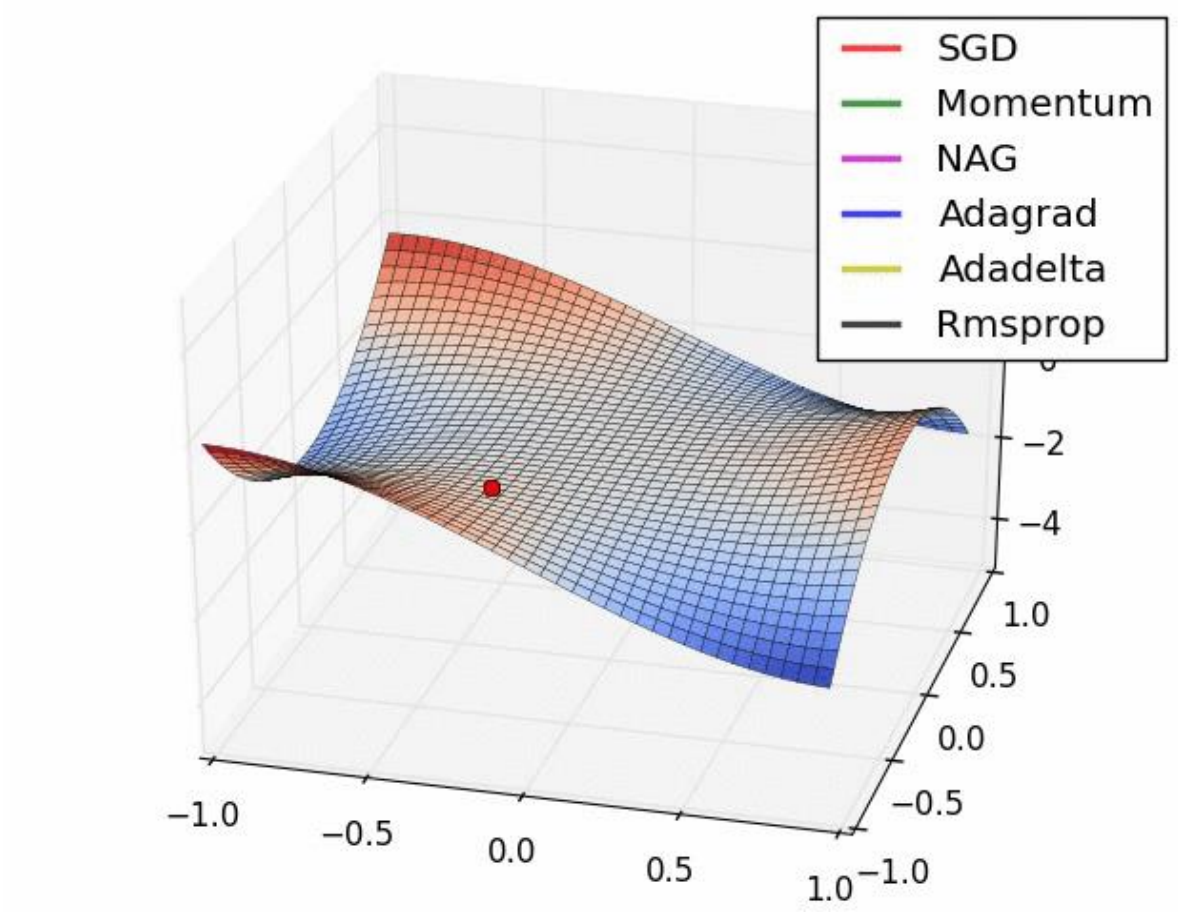
Error Surface





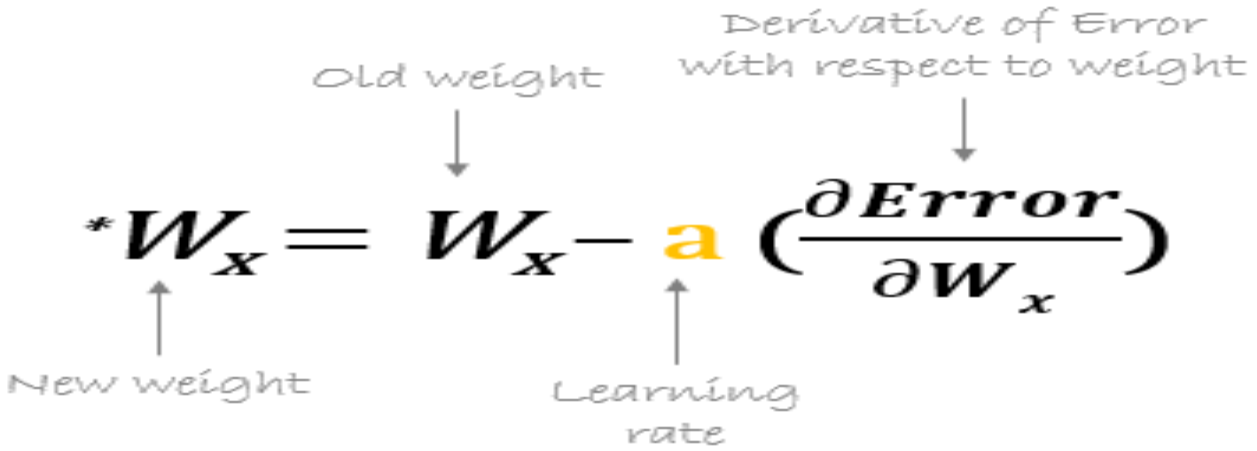
Andrew Ng

Optimizers

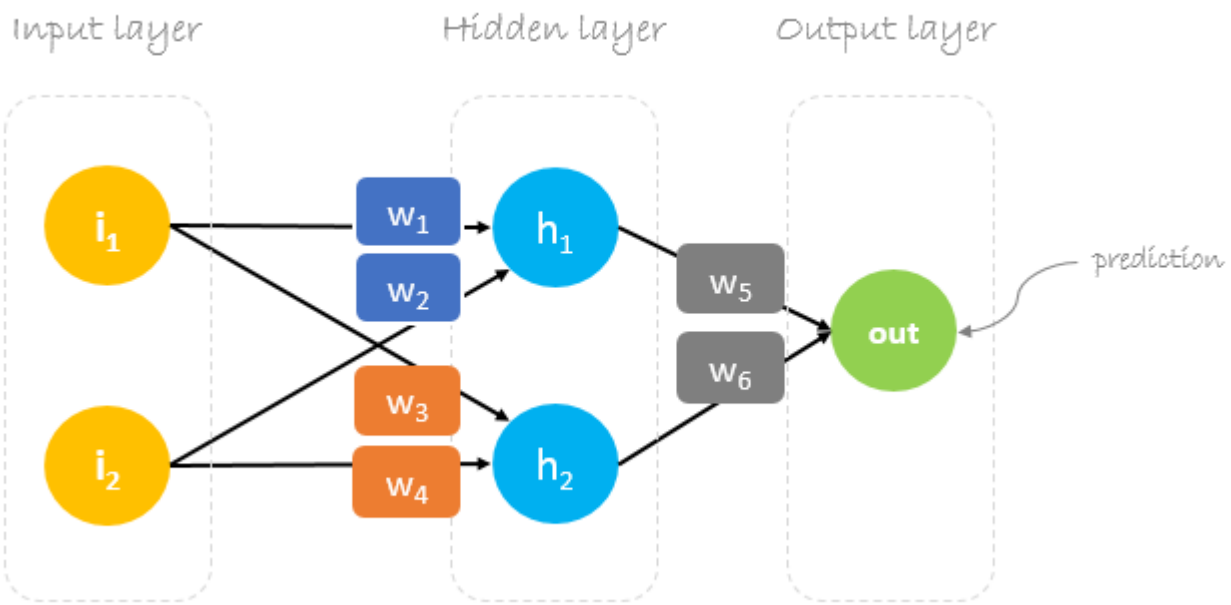

SGD
Momentum
NAG
Adagrad
Adadelta
Rmsprop
−2
−4
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0

Back Propagation

- “Backward propagation of errors”, is a mechanism used to update the weights using gradient descent.
- Gradient descent is an iterative optimization algorithm to minimize the error function.
- The weights are updated in the negative direction of the gradient to find a local minimum of a function



Forward Propagation



$$\text{Out} = f((w_5 * h_1) + (w_6 * h_2))$$

$$\text{Out} = f((w_5 * g_2(w_1 * i_1 + w_2 * i_2) + (w_6 * g_2(w_3 * i_1 + w_4 * i_2))$$

What can be varied to minimize error?

Forward Pass - Example

Our Single Sample:

Inputs = [2, 3]

Output = [1]

Our Initial Weights:

w1 = 0.11

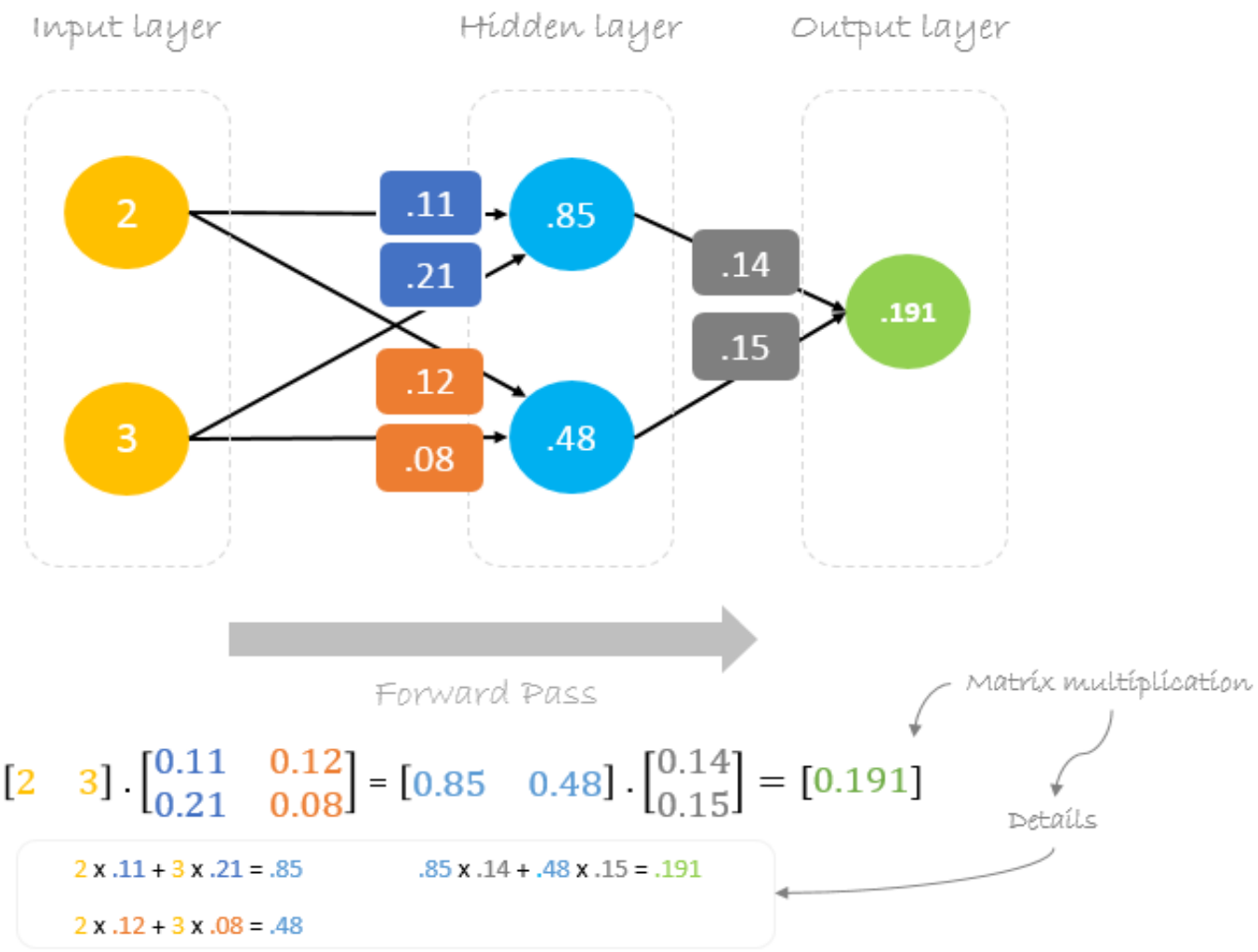
w2 = 0.21

w3 = 0.12

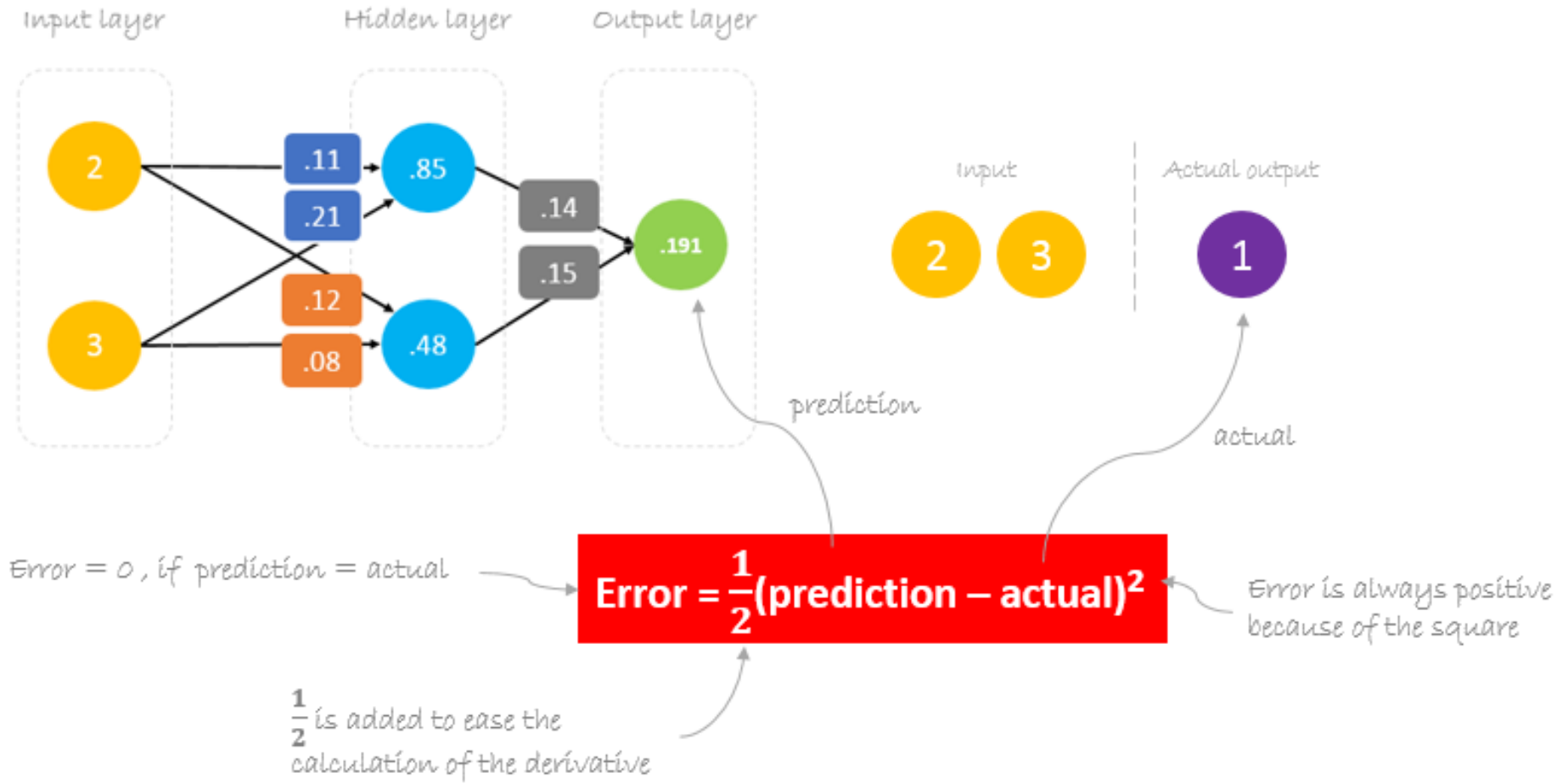
w4 = 0.08

w5 = 0.14

w6 = 0.15



Error Calculation



Error Calculation

- Objective is to ***reduce the Error*** - Difference between Actual & Predicted.

$$\text{prediction} = \text{out}$$

$$\text{prediction} = (h_1)\, w_5 + (h_2)\, w_6$$

$$h_1 = i_1 w_1 + i_2 w_2$$

$$h_2 = i_1 w_3 + i_2 w_4$$

$$\text{prediction} = (i_1 w_1 + i_2 w_2)\, w_5 + (i_1 w_3 + i_2 w_4)\, w_6$$

to change ***prediction*** value,
we need to change ***weights***

Backward Pass – Update W6

$$^{*}W_6 = W_6 - a\left(\frac{\partial Error}{\partial W_6}\right)$$

$$\frac{\partial Error}{\partial W_6} = \frac{\partial Error}{\partial prediction} * \frac{\partial prediction}{\partial W_6}$$

chain rule

Error = $\frac{1}{2}$(prediction – actual)^2

prediction = $(i_1 w_1 + i_2 w_2) w_5 + (i_1 w_3 + i_2 w_4) w_6$

$$\frac{\partial Error}{\partial W_6} = \frac{\frac{1}{2}(predictoin - actula)^2}{\partial prediciton} * \frac{\partial\left(i_1 w_1 + i_2 w_2\right) w_5 + \left(i_1 w_3 + i_2 w_4\right) w_6}{\partial W_6}$$

$$\frac{\partial Error}{\partial W_6} = 2 * \frac{1}{2}(\mathbf{predictoin - actula}) \frac{\partial(predictoin - actula)}{\partial prediciton} * (i_1 w_3 + i_2 w_4)$$

$h_2 = i_1 w_3 + i_2 w_4$

$$\frac{\partial Error}{\partial W_6} = (\mathbf{predictoin - actula}) * (h_2)$$

Δ = prediction – actual

delta

$$\frac{\partial Error}{\partial W_6} = \Delta h_2$$

Backward Pass – Update W1

$$\frac{\partial Error}{\partial W_1} = \frac{\partial Error}{\partial prediction} * \frac{\partial prediction}{\partial h_1} * \frac{\partial h_1}{\partial W_1}$$ ← chain rule

$$\frac{\partial Error}{\partial W_1} = \frac{\partial \frac{1}{2}(predictoin - actula)^2}{\partial prediciton} * \frac{\partial (h_1)\, w_5 + (h_2)\, w_6}{\partial h_1} * \frac{\partial\, i_1 w_1 + i_2 w_2}{\partial w_1}$$

$$\frac{\partial Error}{\partial W_1} = 2 * \frac{1}{2}(\mathbf{predictoin - actula}) \frac{\partial (predictoin - actula)}{\partial prediciton} * (w_5) * (i_1)$$

$$\frac{\partial Error}{\partial W_1} = (\mathbf{predictoin - actula}) * (w_5 i_1)$$

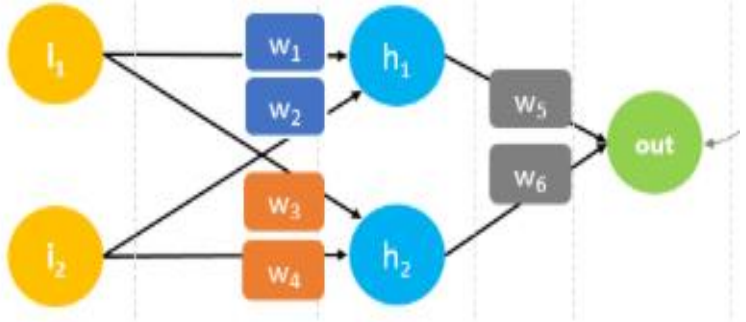
$$\frac{\partial Error}{\partial W_1} = \Delta\, w_5 i_1$$

Error = $\frac{1}{2}$(prediction – actual)²

prediction = $(h_1)\, w_5 + (h_2)\, w_6$

$h_1 = i_1 w_1 + i_2 w_2$

Δ = prediction – actual ← delta



Weight Update through Error propagation

updated weights

$$^{*}w_6 = w_6 - a\,(h_2 \,.\, \Delta)$$

$$^{*}w_5 = w_5 - a\,(h_1 \,.\, \Delta)$$

$$^{*}w_4 = w_4 - a\,(i_2 \,.\, \Delta w_6)$$

$$^{*}w_3 = w_3 - a\,(i_1 \,.\, \Delta w_6)$$

$$^{*}w_2 = w_2 - a\,(i_2 \,.\, \Delta w_5)$$

$$^{*}w_1 = w_1 - a\,(i_1 \,.\, \Delta w_5)$$

$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - a\,\Delta \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} - \begin{bmatrix} a h_1 \Delta \\ a h_2 \Delta \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - a\,\Delta \begin{bmatrix} i_1 \\ i_2 \end{bmatrix} . \begin{bmatrix} w_5 & w_6 \end{bmatrix} = \begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} - \begin{bmatrix} a\, i_1 \Delta w_5 & a\, i_1 \Delta w_6 \\ a\, i_2 \Delta w_5 & a\, i_2 \Delta w_6 \end{bmatrix}$$

Backward Pass - Example

- **Learning rate:** is a Hyperparameter which means that we need to manually guess its value.

$\Delta = 0.191 - 1 = -0.809$ ← **Delta** = prediction - actual

$a = 0.05$ ← **Learning rate,** we smartly guess this number

$$\begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - 0.05(-0.809) \begin{bmatrix} 0.85 \\ 0.48 \end{bmatrix} = \begin{bmatrix} 0.14 \\ 0.15 \end{bmatrix} - \begin{bmatrix} -0.034 \\ -0.019 \end{bmatrix} = \begin{bmatrix} 0.17 \\ 0.17 \end{bmatrix}$$

$$\begin{bmatrix} w_1 & w_3 \\ w_2 & w_4 \end{bmatrix} = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - 0.05(-0.809) \begin{bmatrix} 2 \\ 3 \end{bmatrix} . [0.14 \quad 0.15] = \begin{bmatrix} .11 & .12 \\ .21 & .08 \end{bmatrix} - \begin{bmatrix} -0.011 & -0.012 \\ -0.017 & -0.018 \end{bmatrix} = \begin{bmatrix} .12 & .13 \\ .23 & .10 \end{bmatrix}$$

Back Propagation Algorithm

Algorithm: back_propagation$(h_1, h_2, ..., h_{L-1}, a_1, a_2, ..., a_L, y, \hat{y})$

//Compute output gradient ;

$\nabla_{a_L} \mathscr{L}(\theta) = -(e(y) - \hat{y})$;

for $k = L$ *to* 1 **do**

  // Compute gradients w.r.t. parameters ;

  $\nabla_{W_k} \mathscr{L}(\theta) = \nabla_{a_k} \mathscr{L}(\theta) h_{k-1}^T$;

  $\nabla_{b_k} \mathscr{L}(\theta) = \nabla_{a_k} \mathscr{L}(\theta)$;

  // Compute gradients w.r.t. layer below ;

  $\nabla_{h_{k-1}} \mathscr{L}(\theta) = W_k^T (\nabla_{a_k} \mathscr{L}(\theta))$;

  // Compute gradients w.r.t. layer below (pre-activation);

  $\nabla_{a_{k-1}} \mathscr{L}(\theta) = \nabla_{h_{k-1}} \mathscr{L}(\theta) \odot [\ldots, g'(a_{k-1,j}), \ldots]$;

end

Animation

https://hmkcode.com/netflow/

Loss Functions

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

Where $\hat{y}$ is the predicted value and y is the true value

$$\text{Binary cross entropy} = -(y\log(\hat{y}) + (1-y)\log(1-\hat{y}))$$

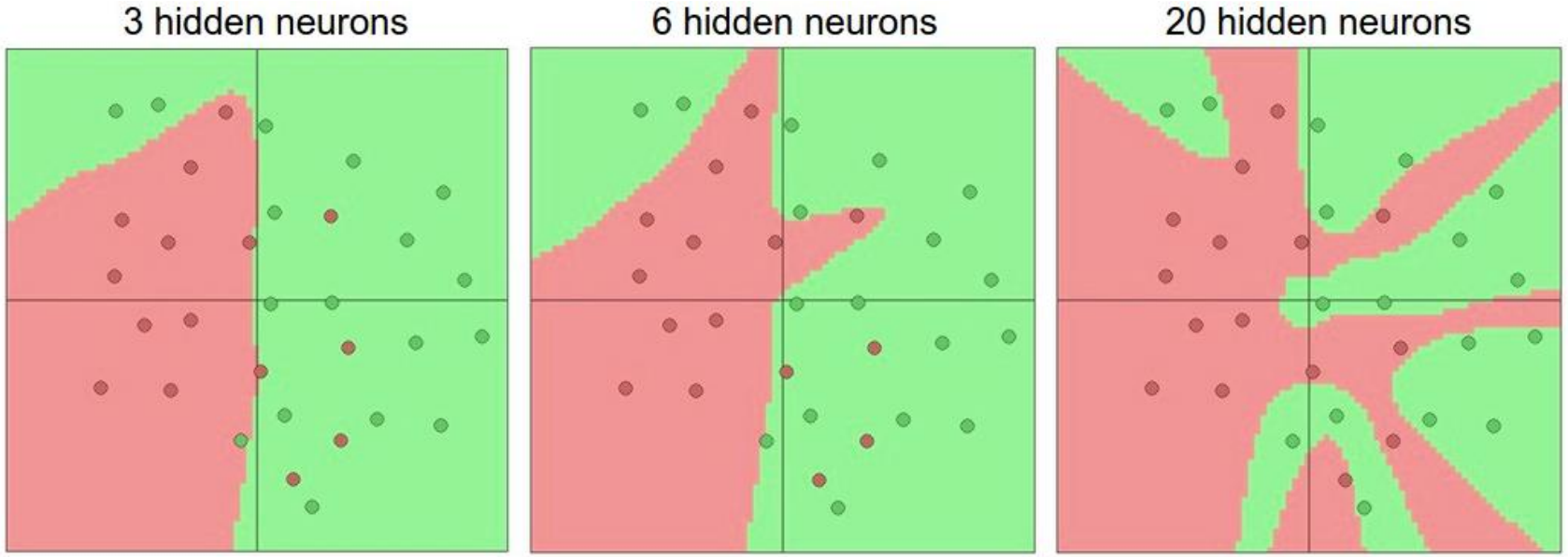
Where $\hat{y}$ is the predicted value and y is the true value

$$\text{Cross entropy} = -\sum_{i}^{M} y_i \log(\hat{y}_i)$$

Where $\hat{y}$ is the predicted value, y is the true value and M is the number of classes

Activation Functions

- Non-linearities needed to learn complex (non-linear) representations of data, otherwise the NN would be just a linear function



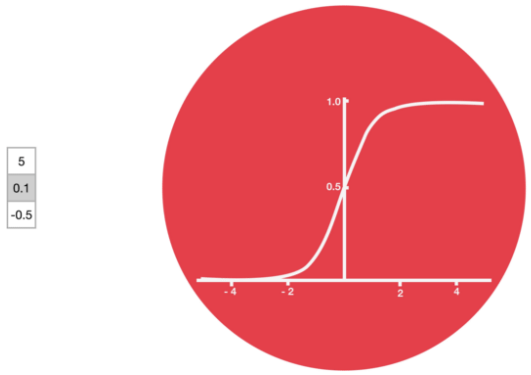
- More layers and neurons can approximate more complex functions

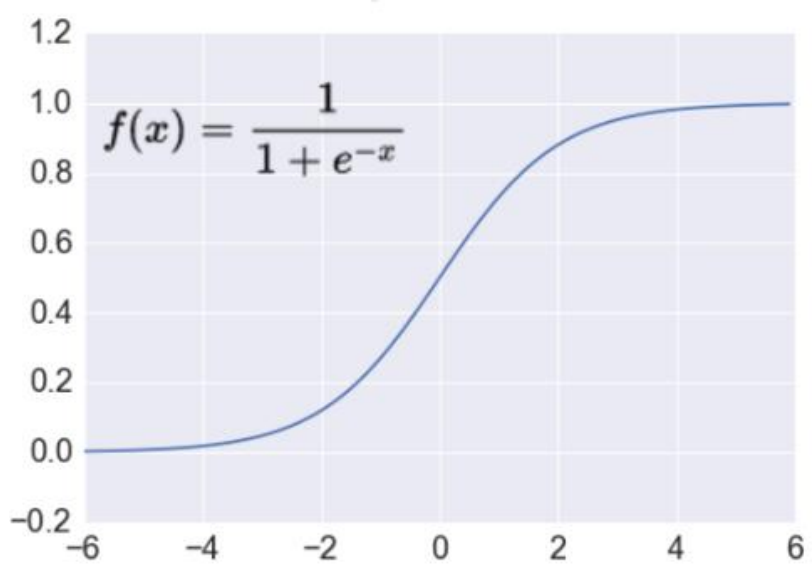
Activation: Sigmoid

- Takes a real-valued number and “squashes” it into range between 0 and 1.

+ Nice interpretation as the **firing rate** of a neuron
  - 0 = not firing at all
  - 1 = fully firing

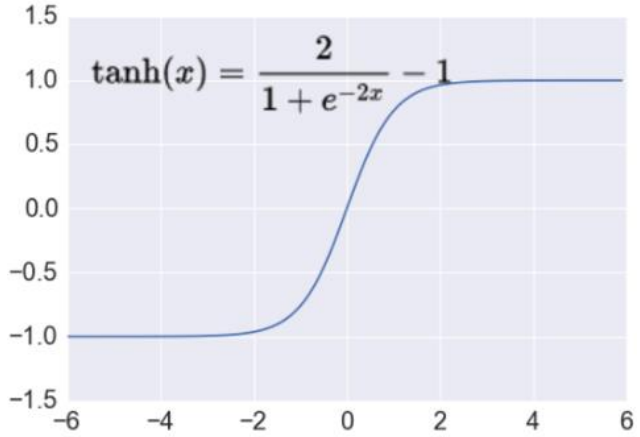
- Sigmoid neurons **saturate** and **kill gradients**, thus NN will barely learn
  - when the neuron’s activation are 0 or 1 (saturate)
    - ☹ gradient at these regions almost zero
    - ☹ almost no signal will flow to its weights
    - ☹ if initial weights are too large then most neurons would saturate

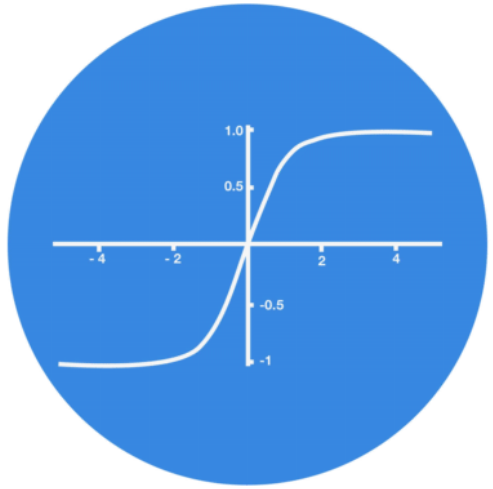




Activation: Tanh

- Takes a real-valued number and “squashes” it into range between -1 and 1.





- Like sigmoid, tanh neurons **saturate**
- Unlike sigmoid, output is **zero-centered**
- Tanh is a **scaled sigmoid**: $\tanh(x) = 2sigm(2x) - 1$

Activation: ReLU

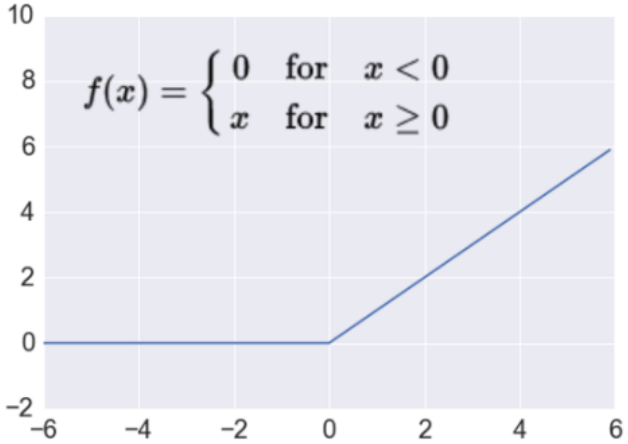
- Takes a real-valued number and thresholds it at zero
- Most Deep Networks use ReLU nowadays

☺ Trains much **faster**
  - accelerates the convergence of SGD
  - due to linear, non-saturating form

☺ Less expensive operations
  - compared to sigmoid/tanh (exponentials etc.)
  - implemented by simply thresholding a matrix at zero

☺ More **expressive**

☺ Prevents the **gradient vanishing problem**



Tips to choose activation and loss functions

Problem Type	Output Type	Final Activation Function	Loss Function
Regression	Numerical value	Linear	Mean Squared Error (MSE)
Classification	Binary outcome	Sigmoid	Binary Cross Entropy
Classification	Single label, multiple classes	Softmax	Cross Entropy
Classification	Multiple labels, multiple classes	Sigmoid	Binary Cross Entropy

Batch Learning

Batch Learning:

- *Adjustment* of the weights of the MLP is *performed after* the presentation of *all* the *N training examples* $\mathcal{T}$.
  - this is called an *epoch* of training.
- Thus, *weight adjustment* is made on *epoch-by-epoch* basis.
- After each epoch, the *examples* in the training samples $\mathcal{T}$ are *randomly shuffled*.

- **Advantages:**
  - *Accurate estimation* of the *gradient vector* (the derivates of the cost function $\mathcal{E}_{av}$ w.r.t. the weight vector **w**), which therefore *guarantee* the *convergence* of the method of steepest descent to a *local minimum*.
  - *Parallelization* of the learning process.
- **Disadvantages:** it is *demanding* in terms of *storage* requirements.

Online Learning

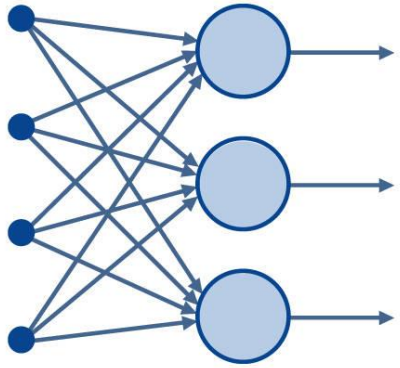
On-line Learning:

- *Adjustment* of the weights of the MLP are *performed* on an *example-by-example* basis.
- The *cost function* to be minimized is therefore the *total instantaneous error* $\mathscr{E}(n)$.
- An *epoch* of training is the presentation all the N samples to the network. Also, in each epoch the *examples* are *randomly shuffled*.

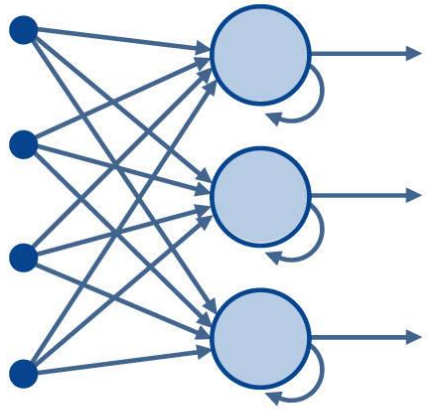
- **Advantages:**
  - Its *stochastic* learning nature, make it ***less likely*** to be *trapped* in *local minimum*.
  - it is *much less demanding* in terms of *storage* requirements.
- **Disadvantages:**
  - We *can not Parallelize* the learning process.

Types of NNs based on Architectures

- The architecture of a Neural Network can be broadly classified into two, namely:
  1. Feed Forward Artificial Neural Network (DNN, CNN..)
  2. Recurrent Neural Network (RNN, LSTM, GRU..)



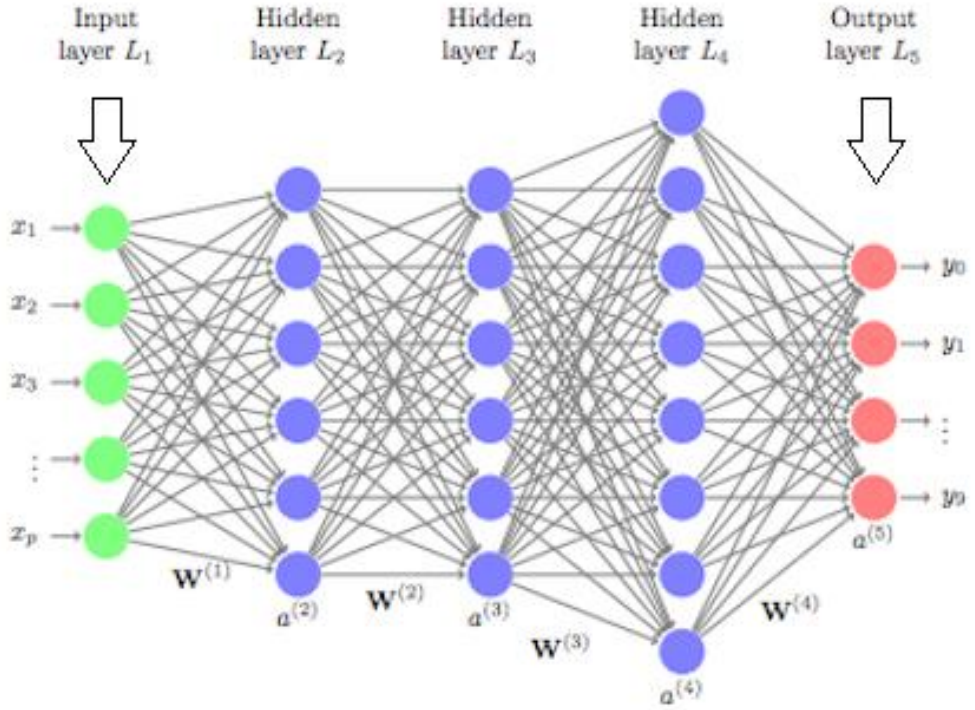
- The information must flow from input to output only in one direction.
- No Feedback loops must be present.



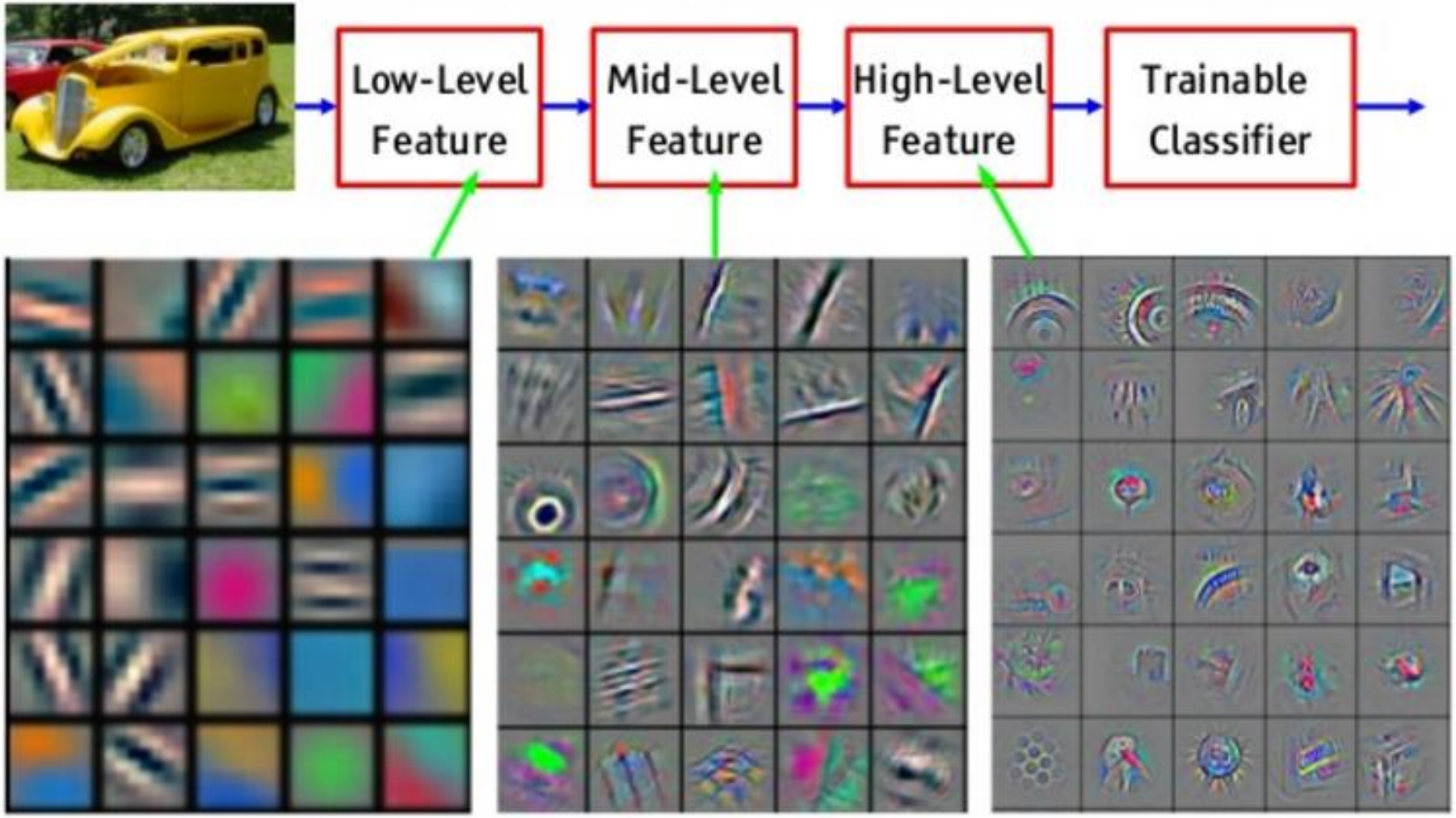
- Information can be transmitted in both directions.
- Feedback loops are allowed.

Deep Neural Network (DNN)

- A Deep Neural Network (DNN) is an Artificial Neural Network (ANN) with multiple layers between the input and output layers.
- The DNN finds the correct mathematical manipulation to turn the input into the output, whether it be a linear relationship or a non-linear relationship.



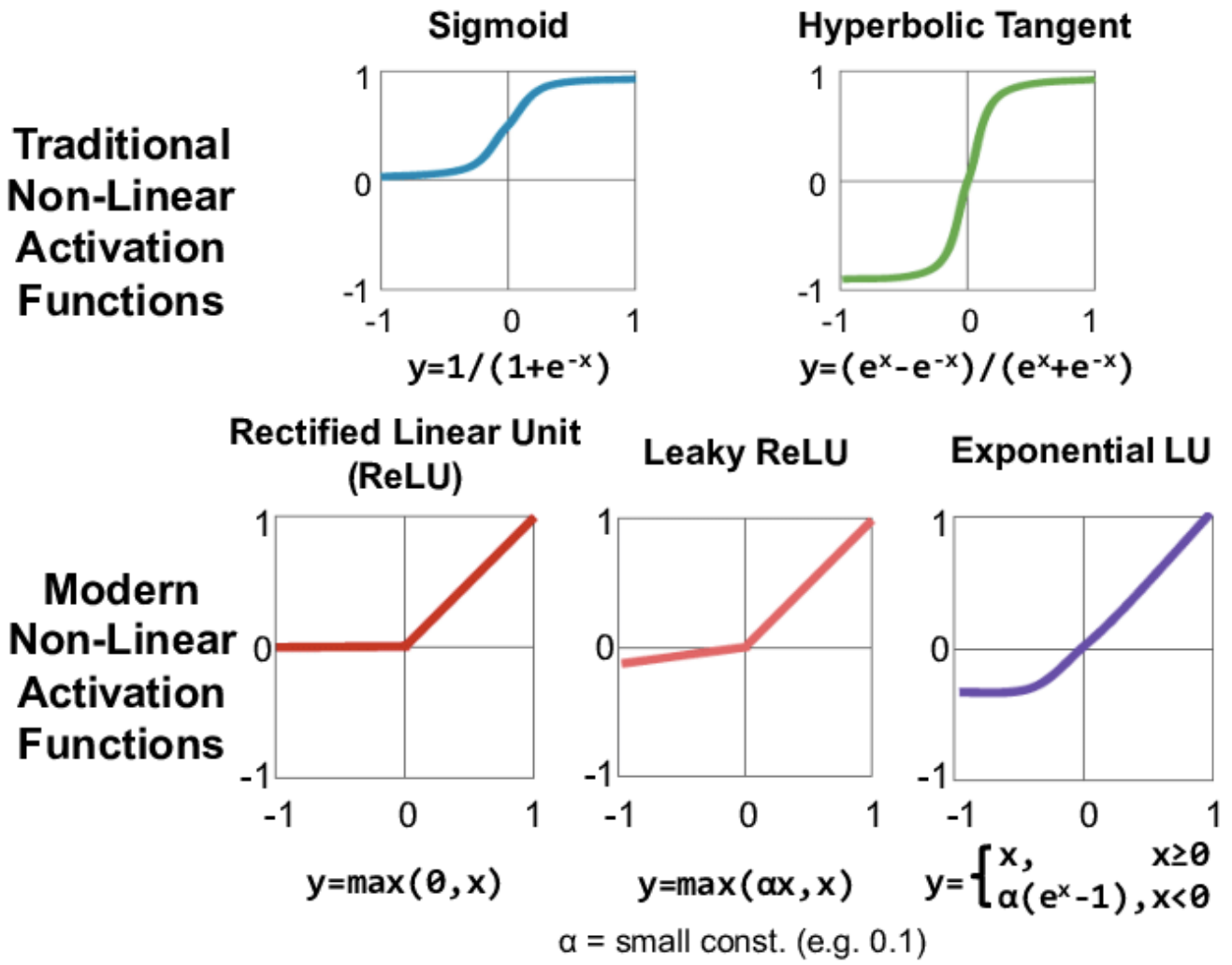
What DNNs Learn…



MLP vs DNN

- Number of Layers
- Datasets (Size)
- Computing Resources (Memory & Time efficient)
- Weight Initialization
- Non-linear activation Functions

MLP vs DNN


Traditional Non-Linear Activation Functions
Sigmoid
y=1/(1+e^-x)
Hyperbolic Tangent
y=(e^x-e^-x)/(e^x+e^-x)
Modern Non-Linear Activation Functions
Rectified Linear Unit (ReLU)
y=max(0,x)
Leaky ReLU
y=max(αx,x)
Exponential LU
y={x, x≥0; α(e^x-1), x<0
α = small const. (e.g. 0.1)

Implementing DNN in Python

Convolutional Neural Network…

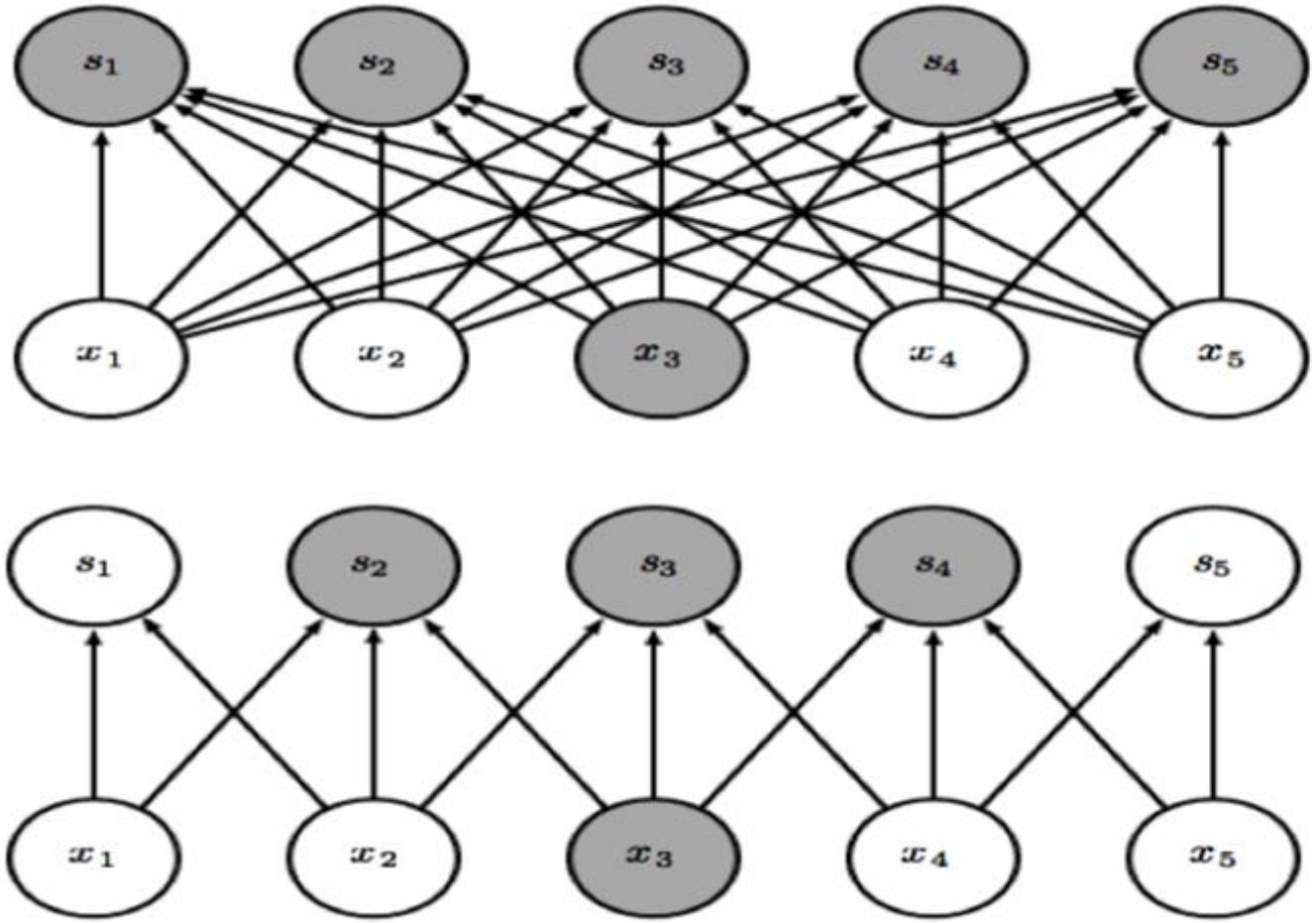
- Inspired by the human visual system
- Convolutional Neural Networks is an extension of traditional Multi-layer Perceptron, based on 3 ideas:
  1. Local receptive fields
  2. Shared weights
  3. Spatial sub-sampling

- See LeCun paper (1998) on text recognition:
- http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf

Convolutional Neural Network (ConvNet)

- A class of deep, feed-forward neural networks
- Learns a complex representation of **visual** data using vast amounts of data.
- Learns multiple layers of transformations, which are applied on top of each other to extract a progressively **more sophisticated representation** of the input.
- CNN - multi-layer NN architecture
  - Convolutional + Non-Linear Layer
  - Sub-sampling Layer
  - Fully connected layers

Dense vs Sparse Layers



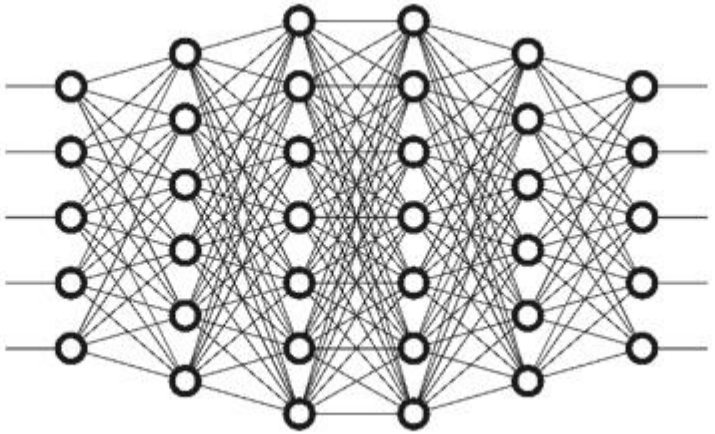
Dense

Sparse

Why CNN…?

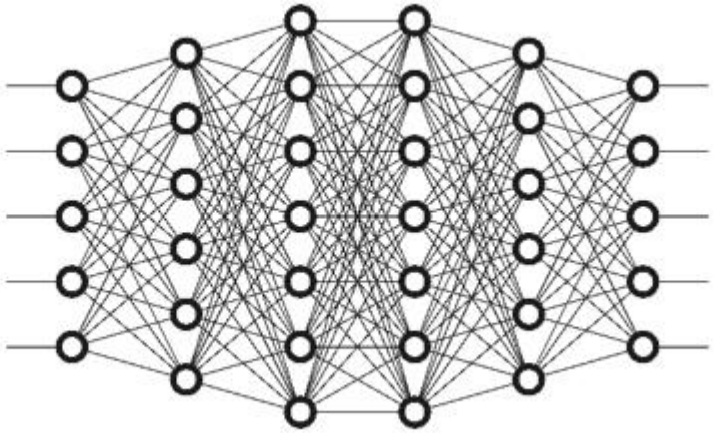
- Images are high-dimensional usually 112*196.
- It would take a huge amount of parameters to characterize the network.

Image with 28 x 28 x 3 pixels



Number of weights in the first hidden layer will be 2352

Image with 200 x 200 x 3 pixels



Number of weights in the first hidden layer will be 120,000

Layers of CNN

- CNN has multiple hidden layers that help in extracting information from an image.

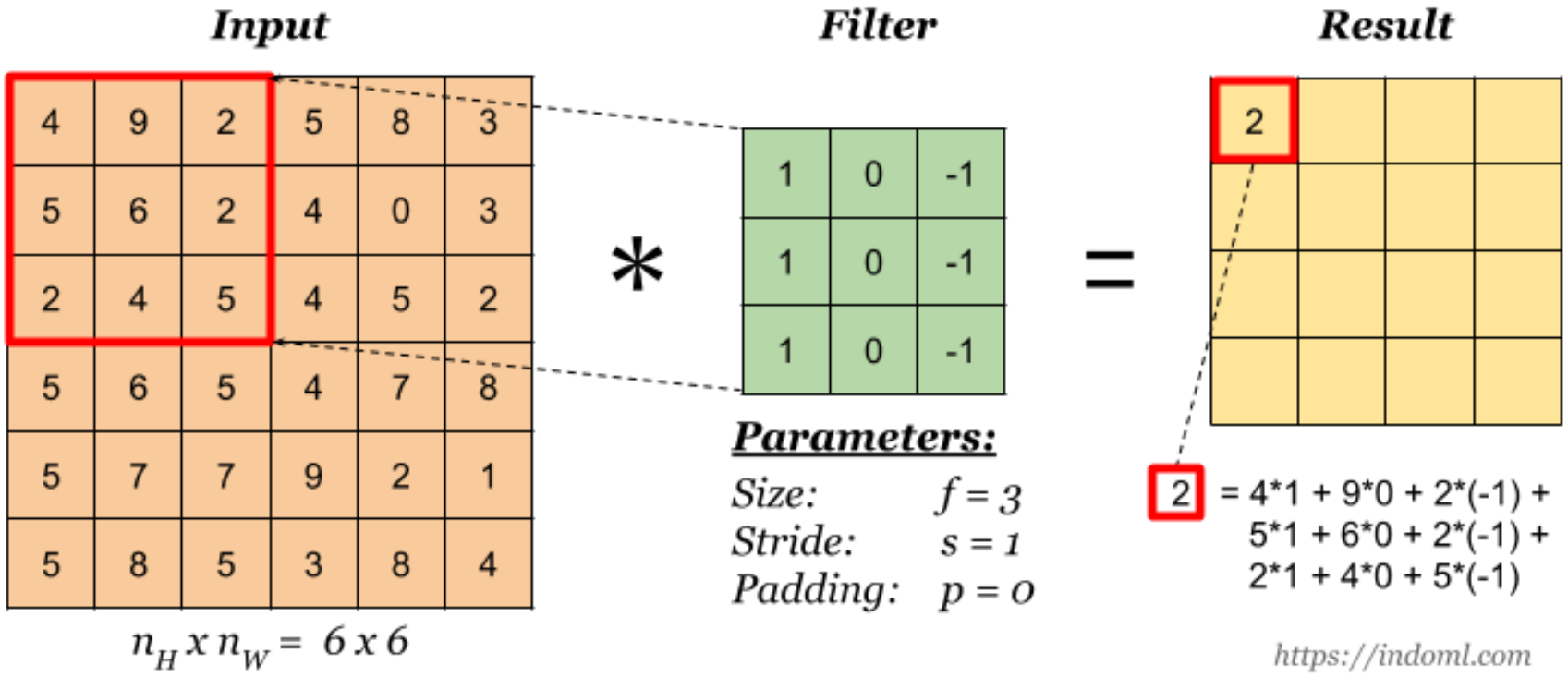
1. Convolution layer
2. Activation layer
3. Subsampling / Pooling layer
4. Fully connected layer

Intuition behind Convolutions

- Different edge detection filters applied on input Image

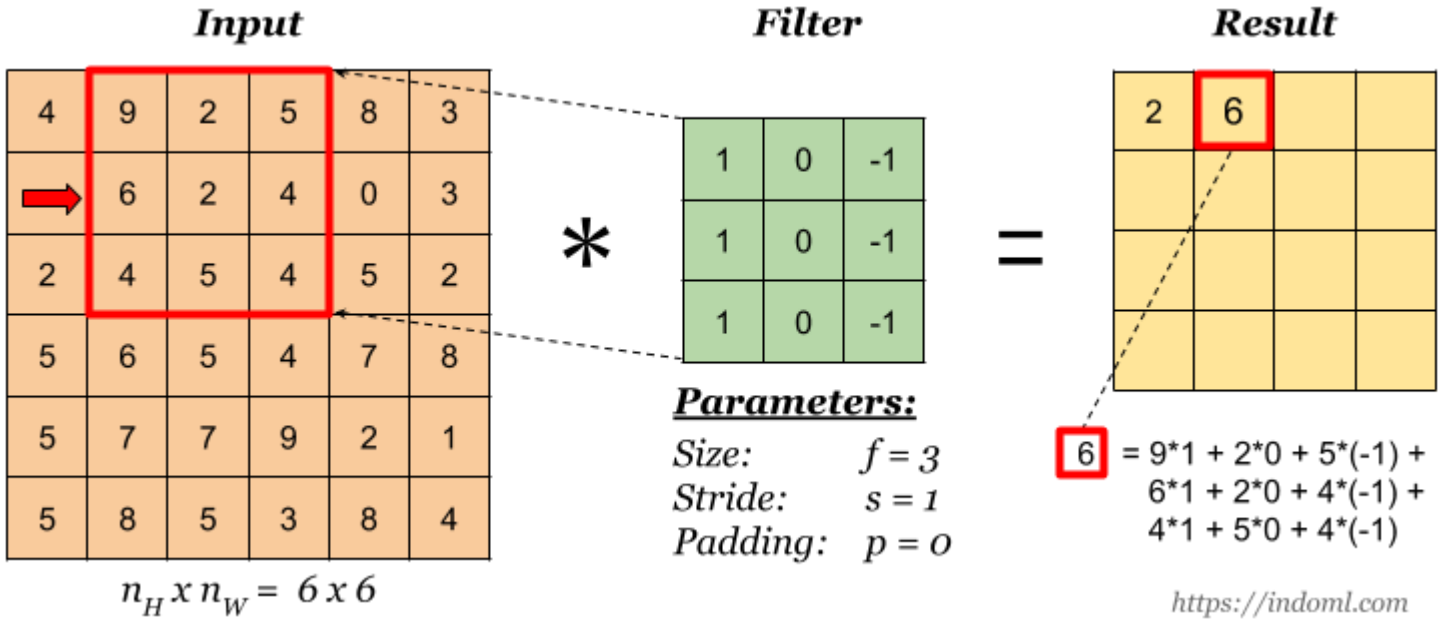
Convolution Operation

Step-1: Overlay the filter to the input, perform element wise multiplication, and add the result.



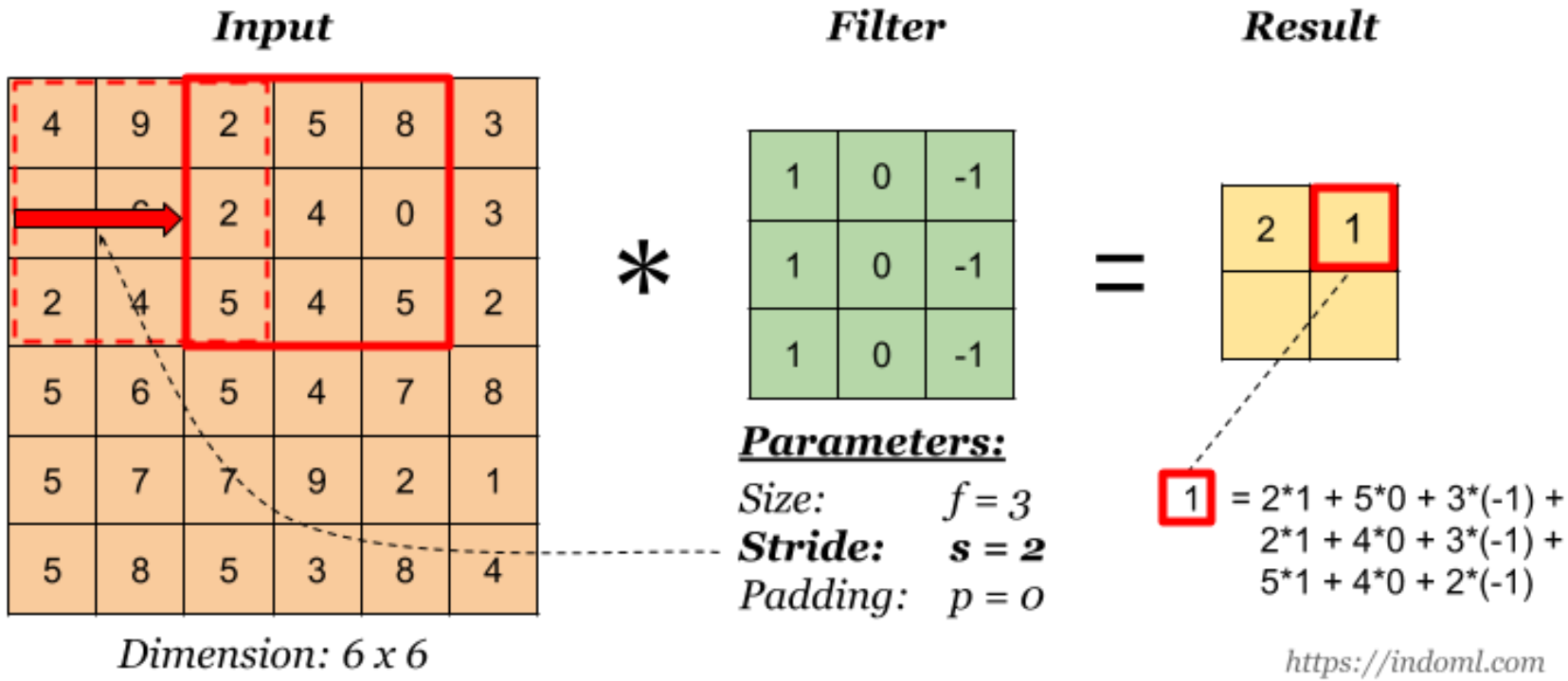
Convolution Operation

Step-2: Move the overlay right one position (or according to the stride setting), and do the same calculation in step1. Repeat this.



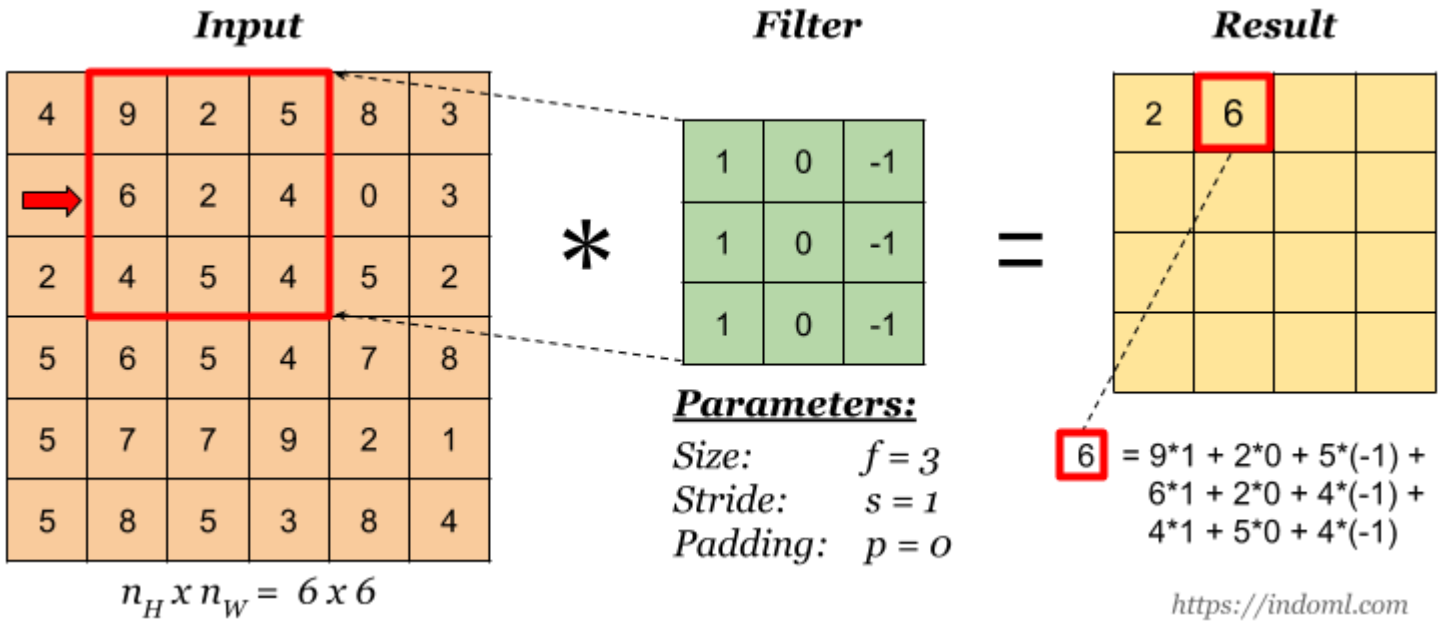
Convolution Operation: Stride

- Stride governs how many cells the filter is moved in the input to calculate the next cell in the result.



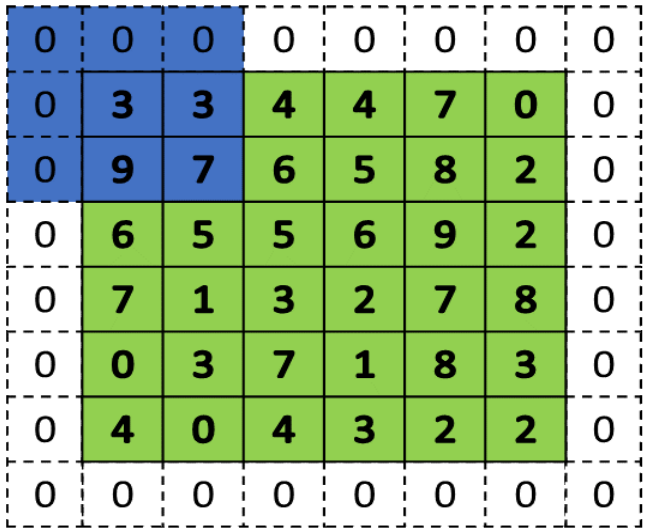
Convolution Operation: No Padding

- The conv operation without padding results in the output smaller in size compared to the input
- It is not possible to apply convolution on the borders of the image



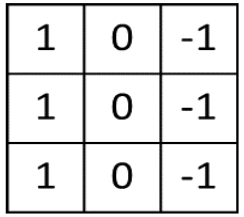
Convolution Operation: Padding

- Allows to retain the input size after convolution operation.
- Allows to extract information from the borders.
- As the size is retained allows more deeper layers



$6 \times 6 \rightarrow 8 \times 8$

$*$



3×3

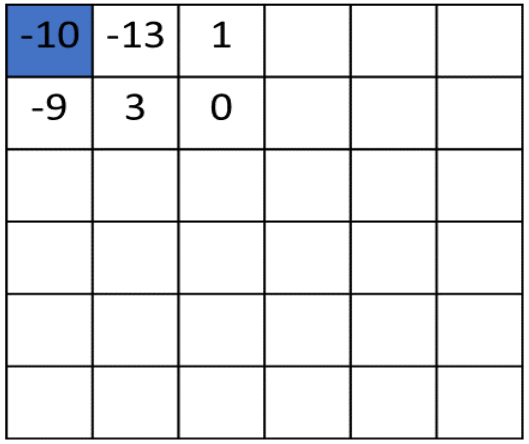
Parameters:

Size: $f = 3$

Stride: $s = 1$

Padding: $p = 1$

$=$



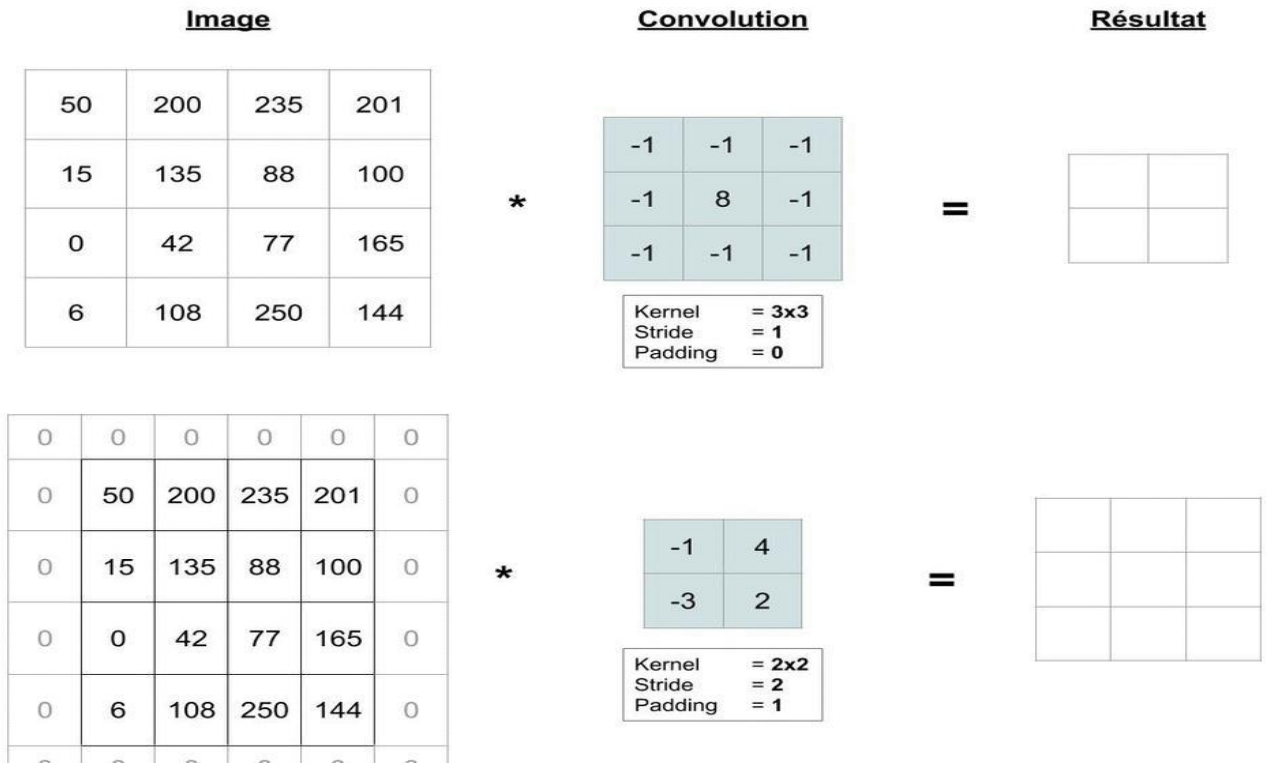
6×6

Convolution Operation

- How to determine output dimension of the convolution layer..?

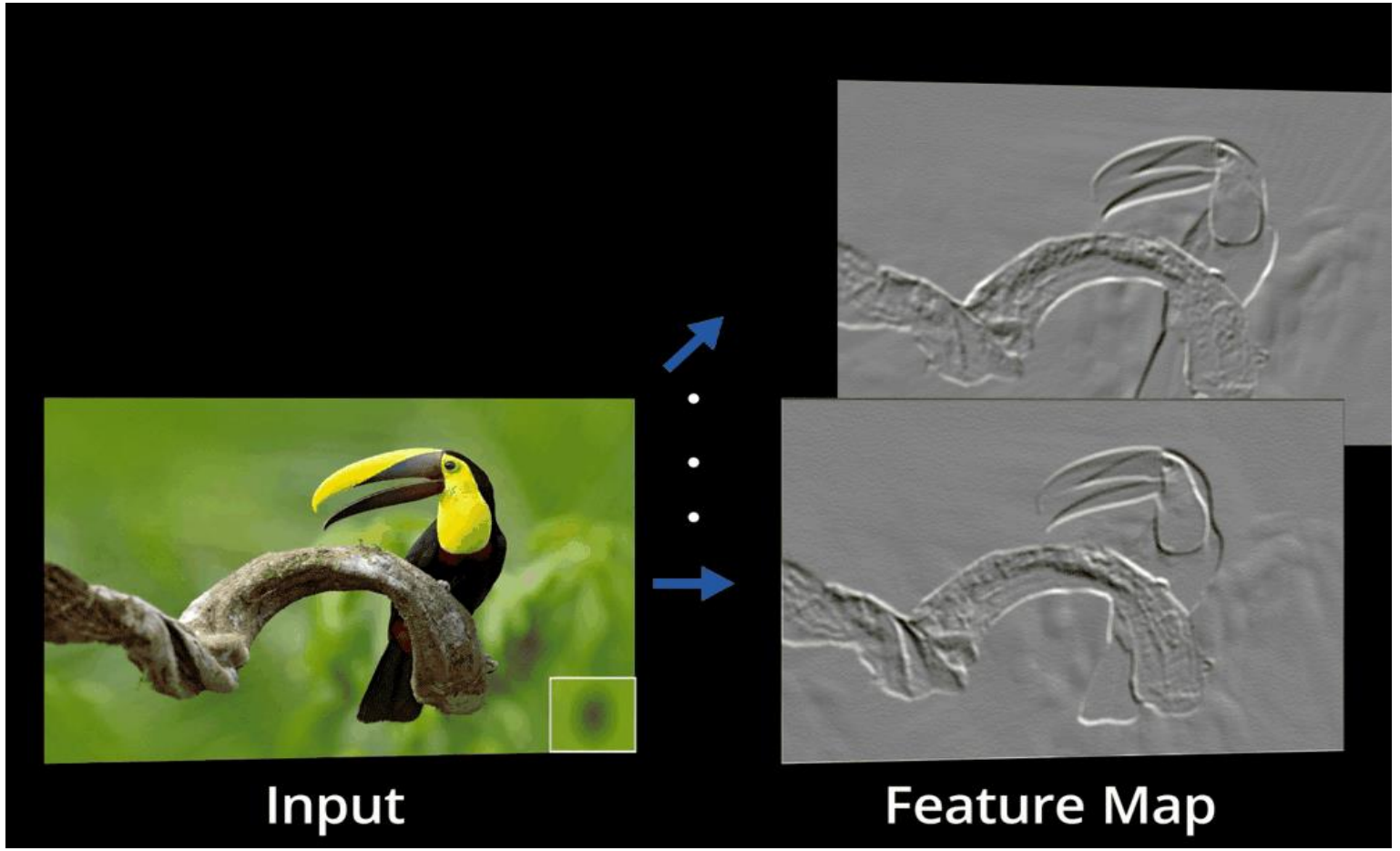
$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

n_{in}: number of input features
n_{out}: number of output features
k: convolution kernel size
p: convolution padding size
s: convolution stride size

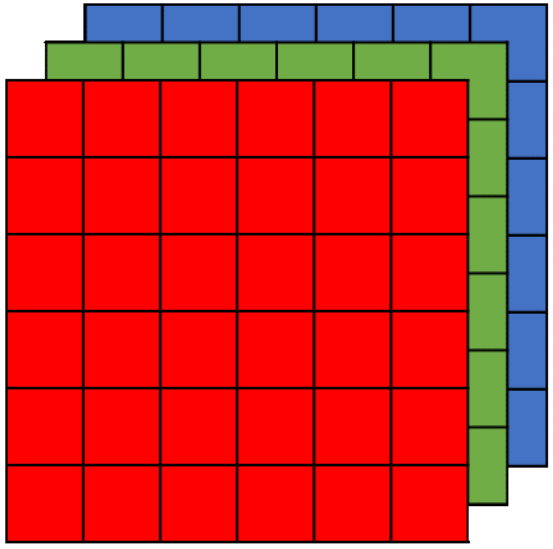


Convolution Operation: Multiple filters

- Each filter learns different sets of features

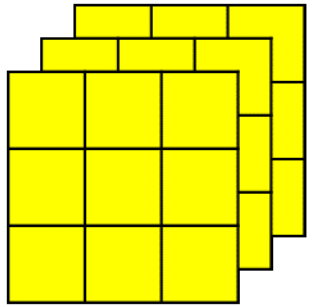


Convolution Operation: RGB channels



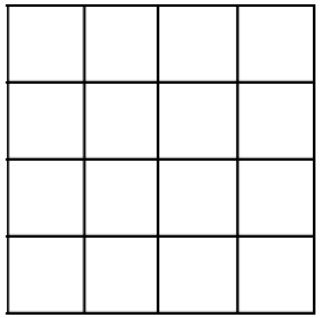
$6 \times 6 \times 3$

*



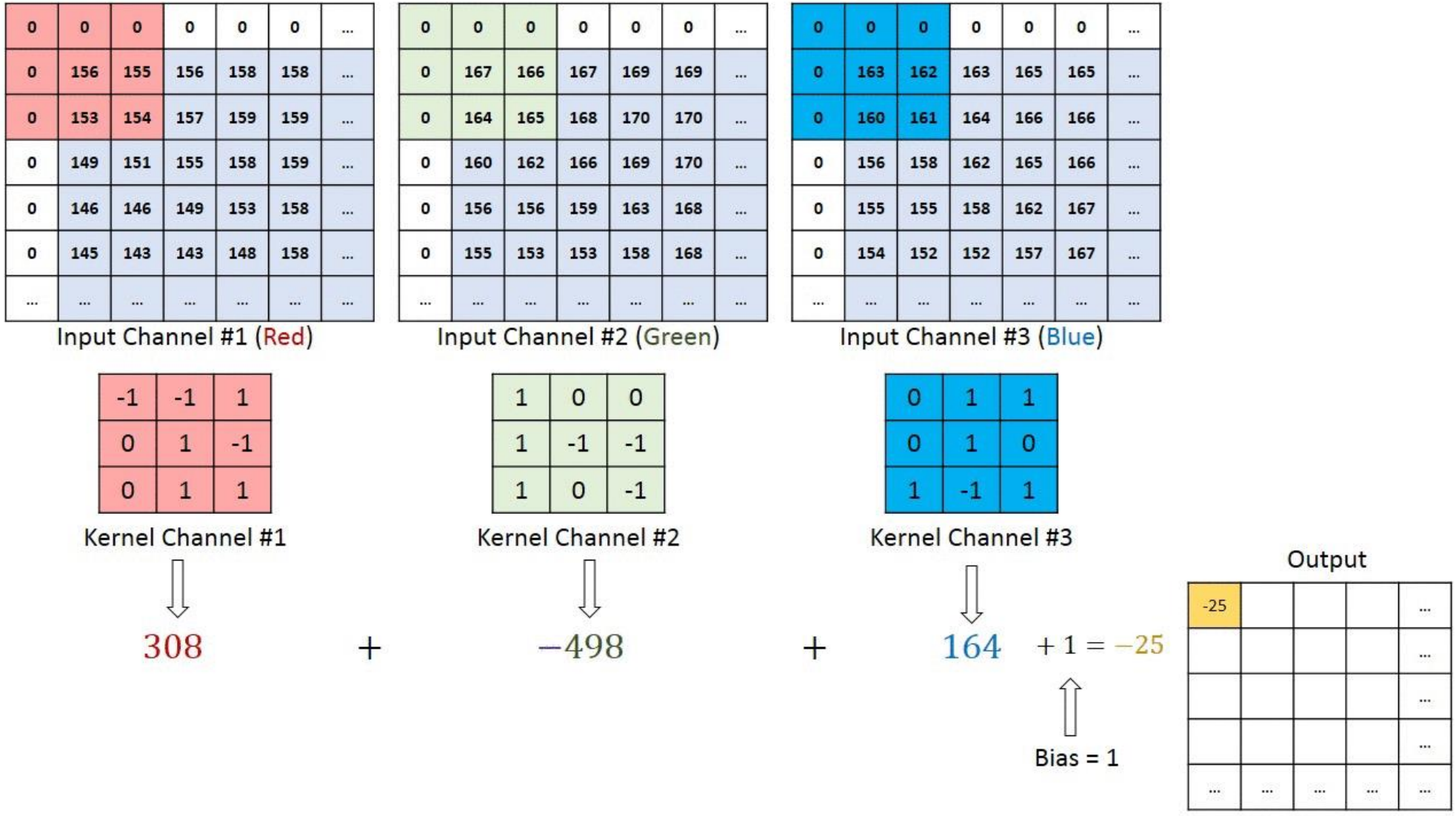
$3 \times 3 \times 3$

=

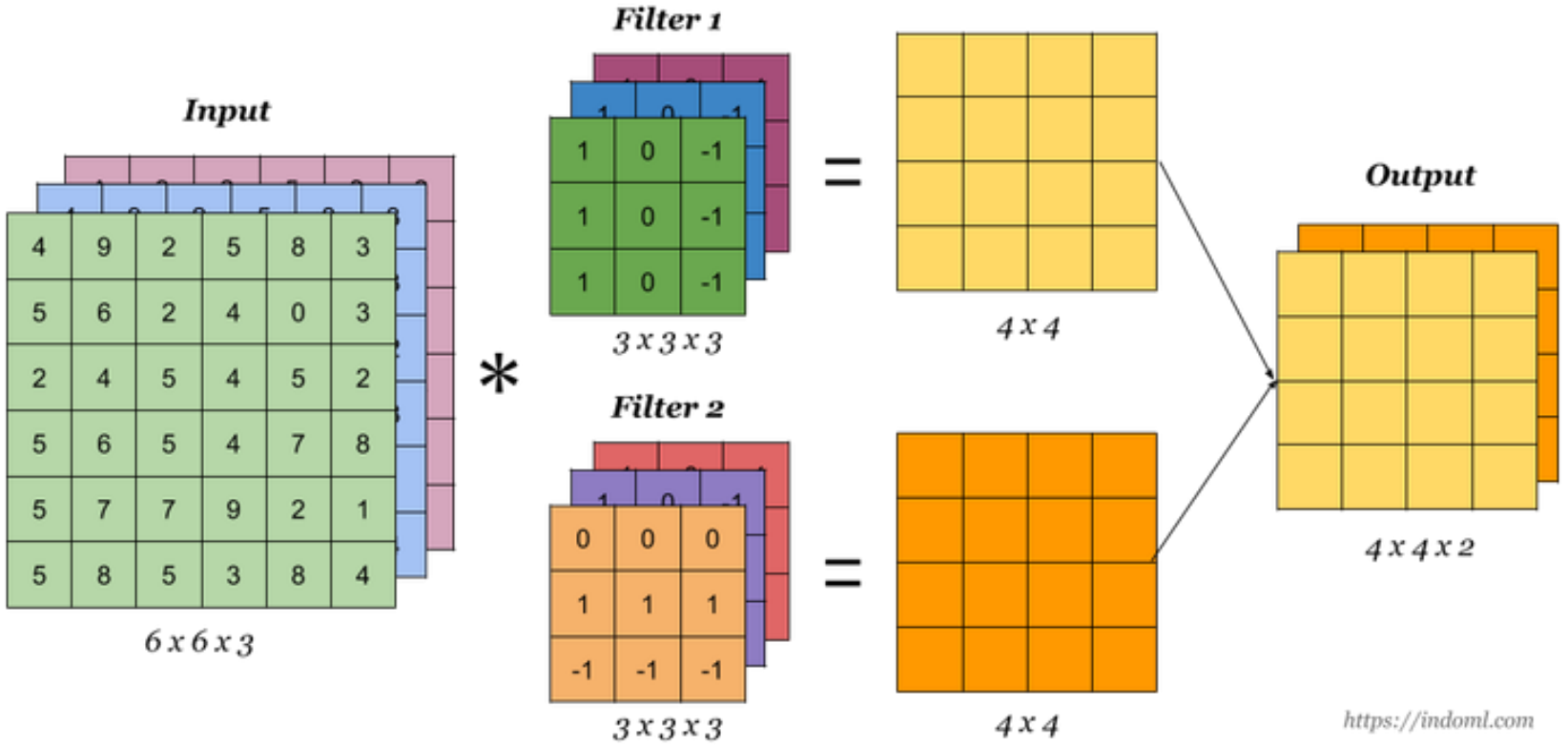


4×4

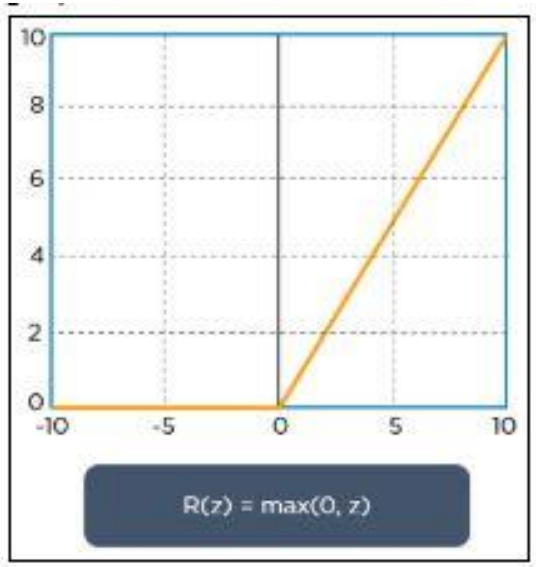
Convolution Operation: RGB channels



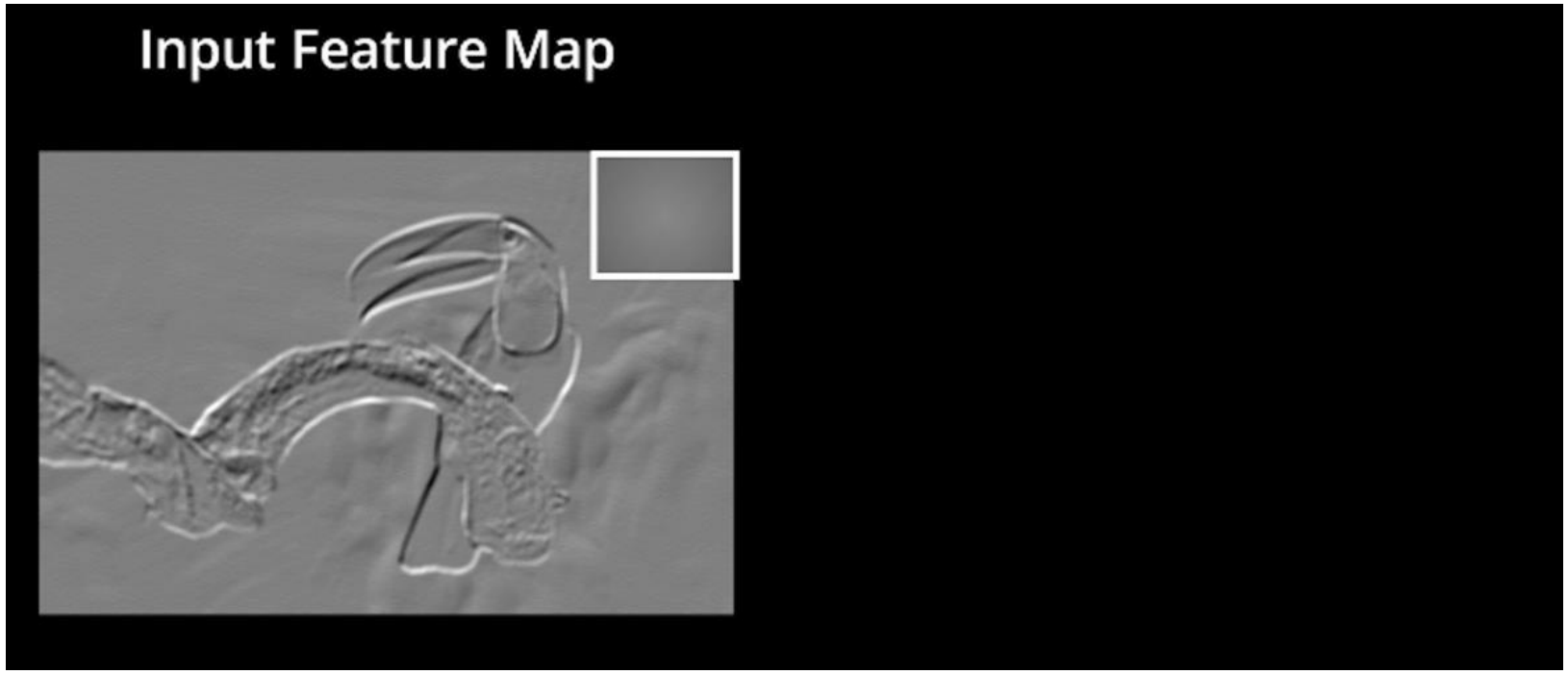
Convolution Operation: RGB Multiple filters



Convolution + Activation Operation



ReLU

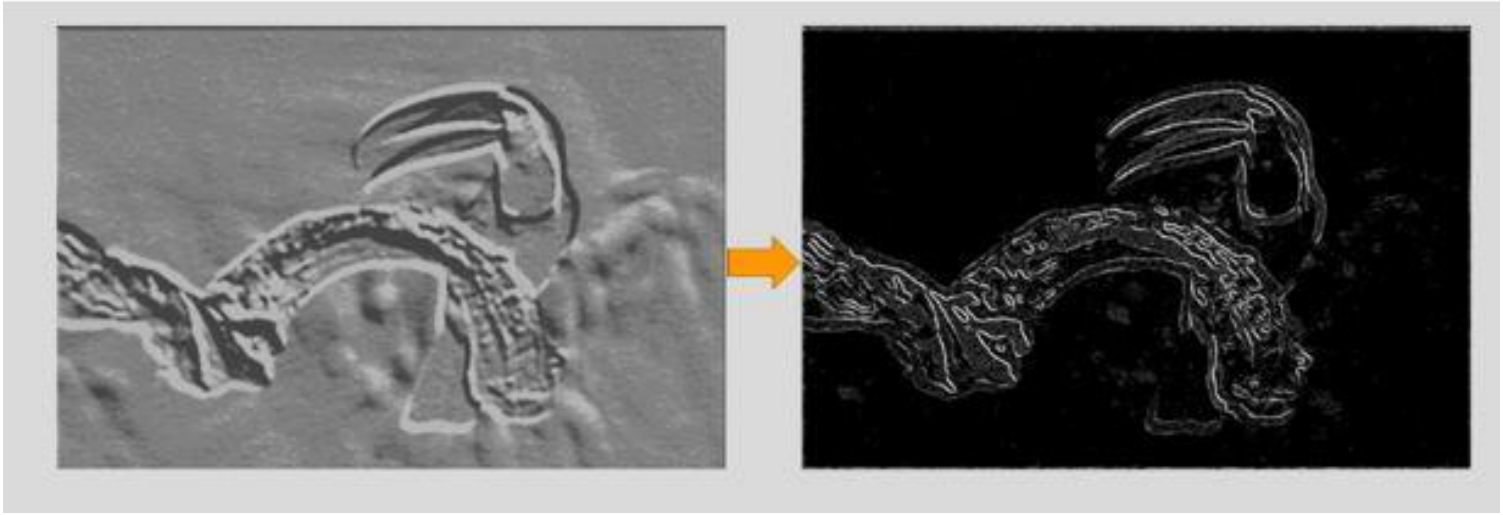


Pooling Operation

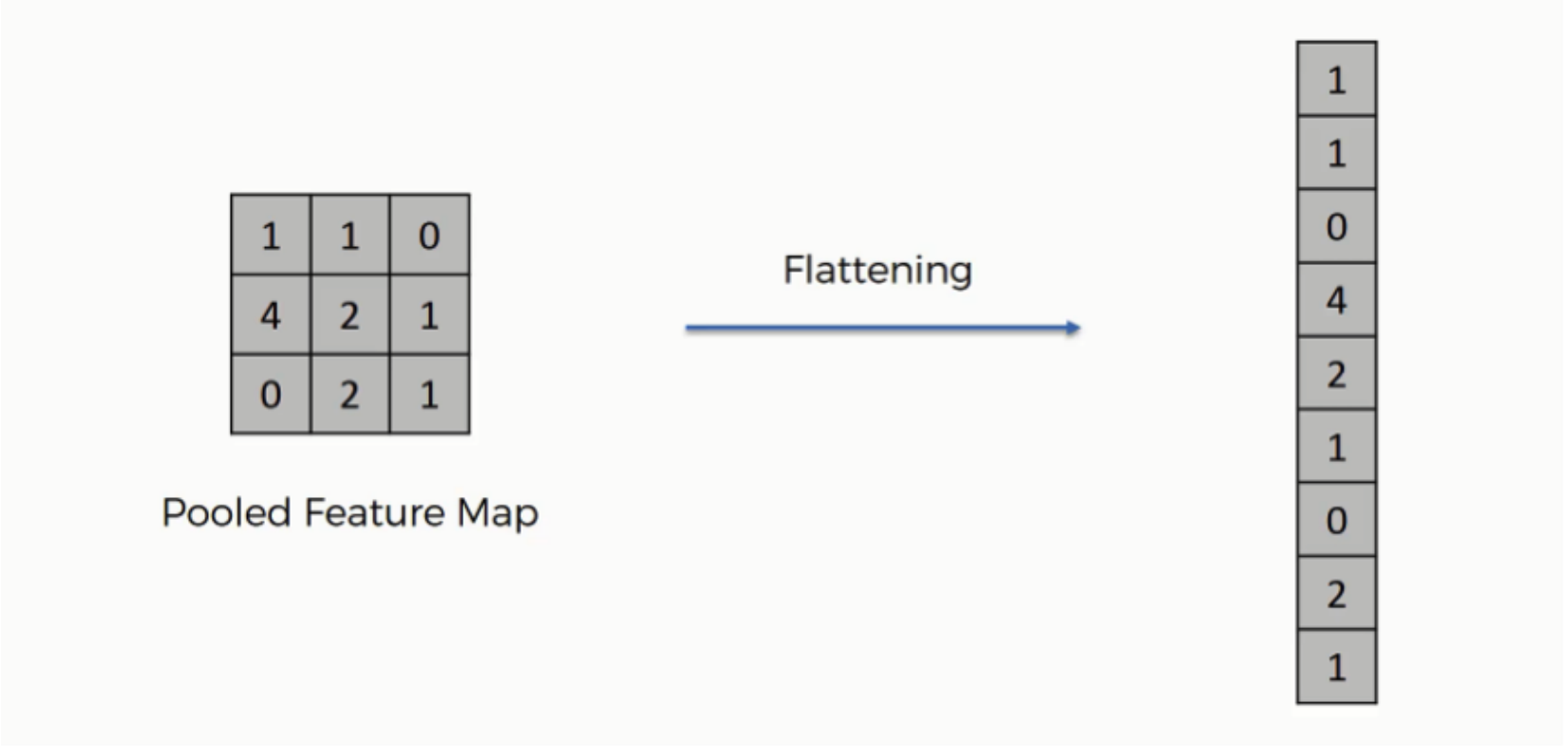
Feature Map

6	6	6	6
4	5	5	4
2	4	4	2
2	4	4	2

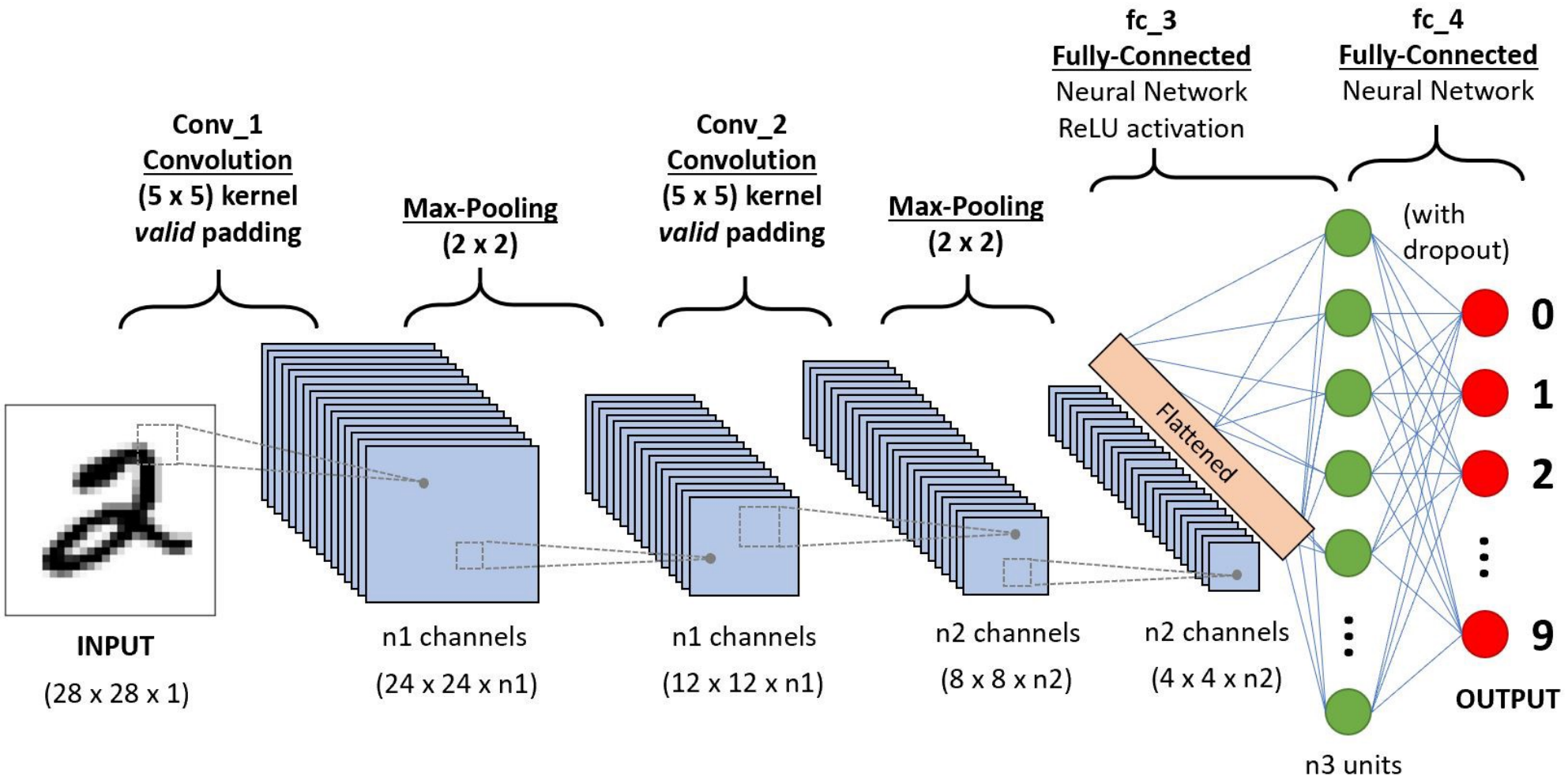
Max Pooling	Average Pooling	Sum Pooling



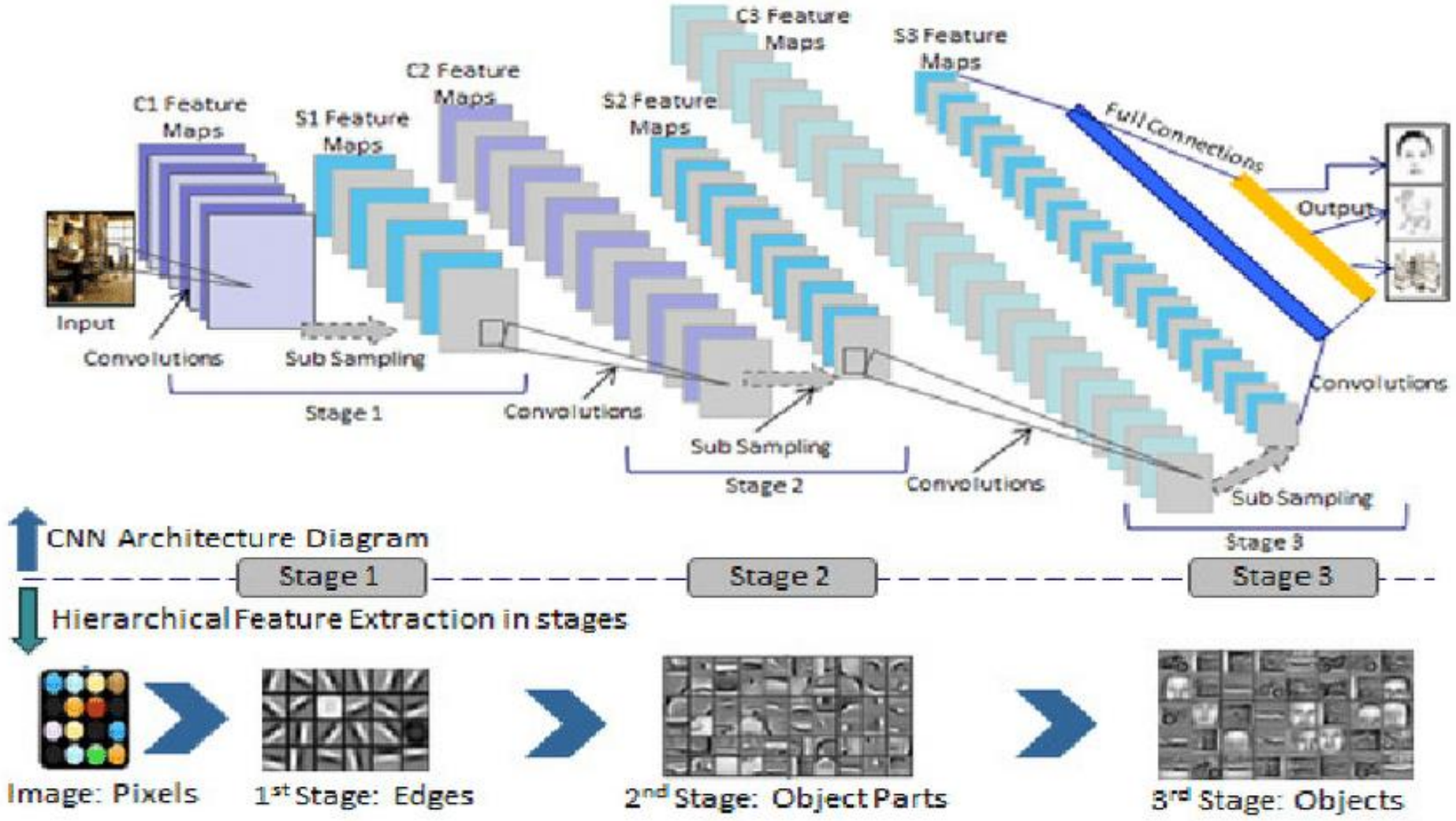
Flatten layer



Sample Architecture of CNN



What CNN Learns…?

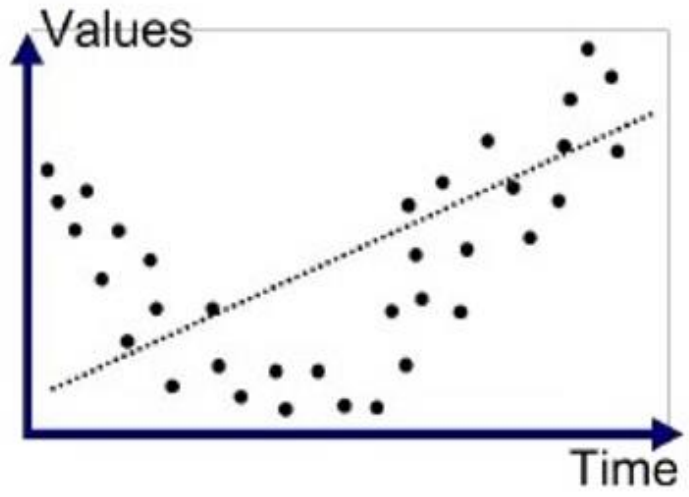


Implementing CNN in Python

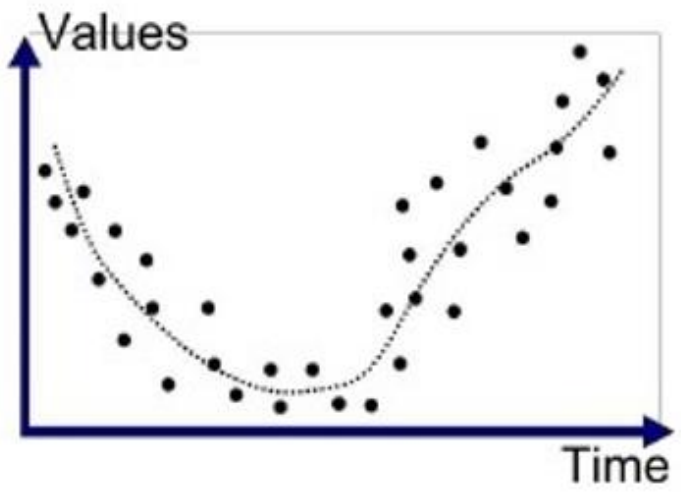
Problem With Overfitting

- Large neural nets trained on relatively small datasets can overfit the training data.
- This has the effect of the model learning the statistical noise in the training data, which results in poor performance when the model is evaluated on new data, e.g. a test dataset.
- Generalization error increases due to overfitting.

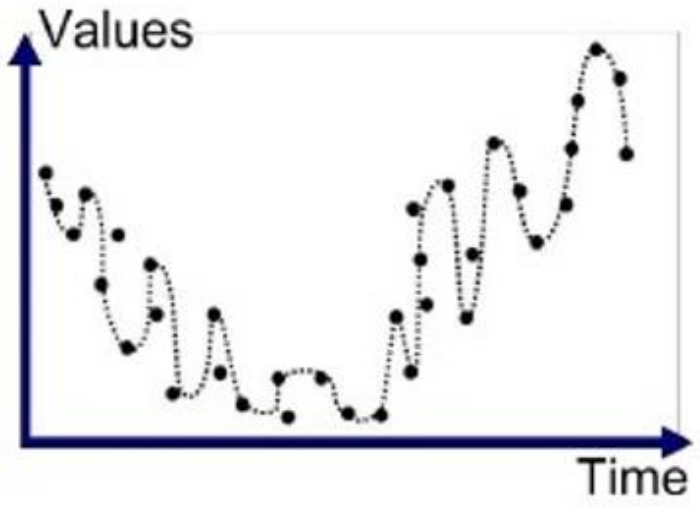
Problem With Overfitting



Underfitted



Good Fit/Robust

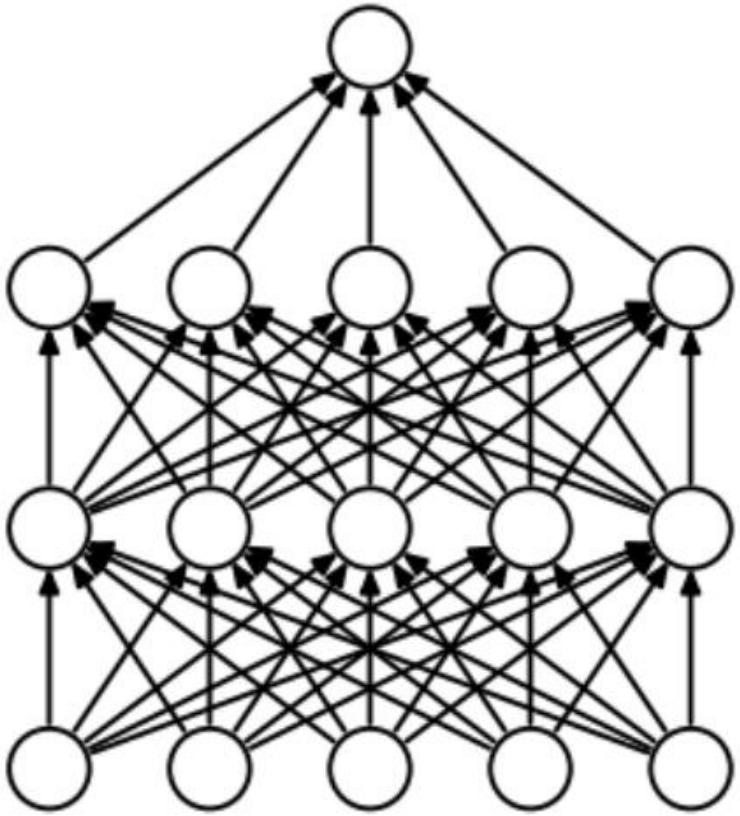


Overfitted

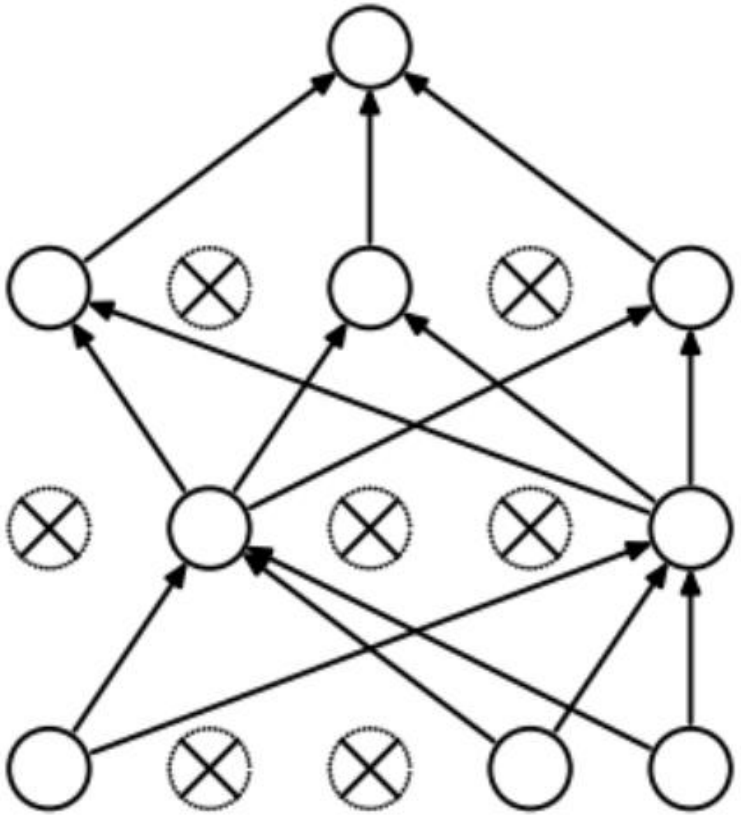
Role of Dropout

- Randomly Drop Nodes
- Dropout is a regularization method that approximates training a large number of neural networks with different architectures in parallel.
- During training, some number of layer outputs are randomly ignored or “dropped out.”
- By dropping a unit out, we mean temporarily removing it from the network, along with all its incoming and outgoing connections.
- Dropout simulates a sparse activation from a given layer, which interestingly, in turn, encourages the network to actually learn a sparse representation as a side-effect.
- As such, it may be used as an alternative to activity regularization for encouraging sparse representation
- Dropout: A Simple Way to Prevent Neural Networks from Overfitting, 2014.

Role of Dropout



(a) Standard Neural Net

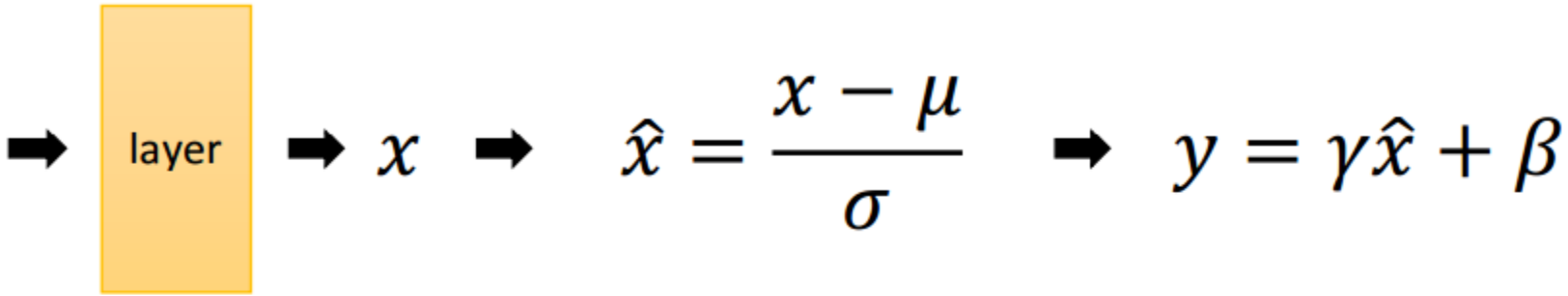


(b) After applying dropout.

Batch Normalization

- Training deep neural networks, e.g. networks with tens of hidden layers, is challenging.
- Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change.
- This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities.
- Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, 2015.
- The authors of the paper introducing batch normalization refer to change in the distribution of inputs during training as "internal covariate shift."

Batch Normalization



- μ: mean of x in mini-batch
- σ: std of x in mini-batch
- γ: scale
- β: shift

- μ, σ: functions of x, analogous to responses
- γ, β: parameters to be learned, analogous to weights

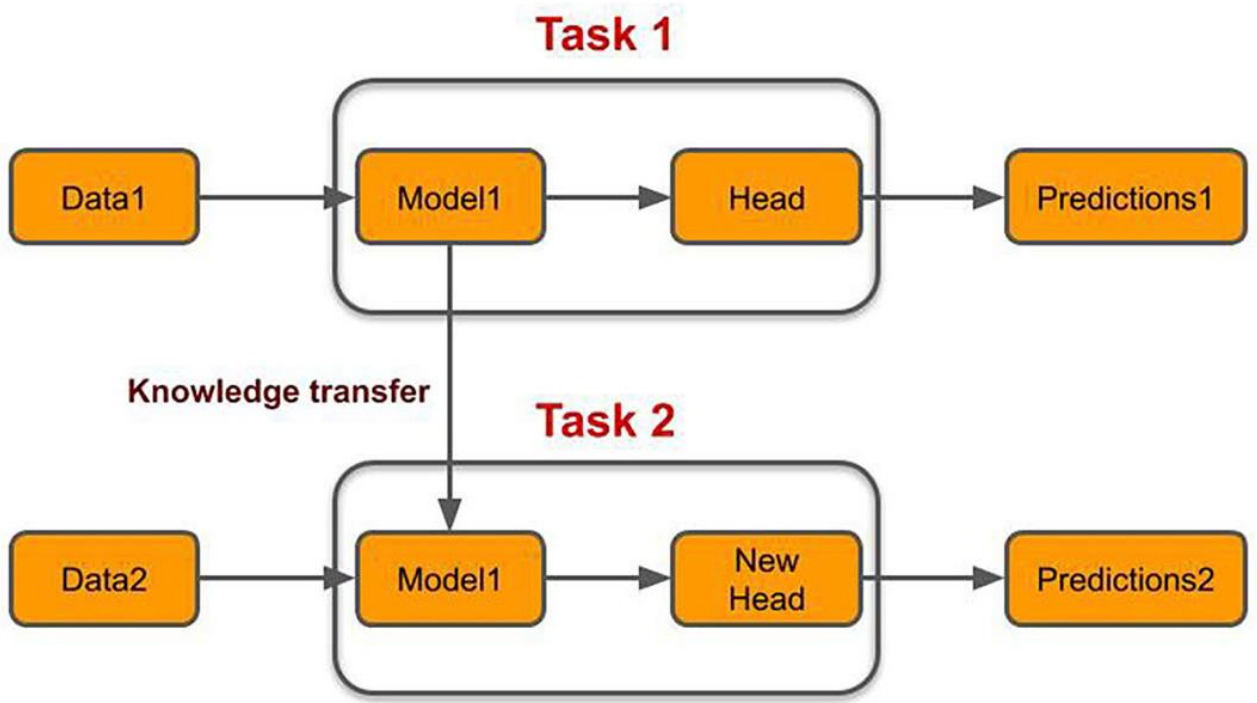
Problem with Deep Neural Networks

- The downside of the Deep / Convolutional Neural Networks (CNN) is that they need enormous amounts of data for training.
- This is usually scarce for most of the real-time applications.
- This problem can be addressed by using Transfer Learning.
- Traditionally Data Augmentation Techniques were also used to increase the size of training data and then train the models for desired task.
- Transfer Learning shows effective results when in comparison with data augmentation techniques.

Implementing Regularized NNs

Transfer Learning…

- Transfer learning is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned.



Approaches

- Two common approaches are as follows:
  1. Develop Model Approach
  2. Pre-trained Model Approach

Develop Model Approach

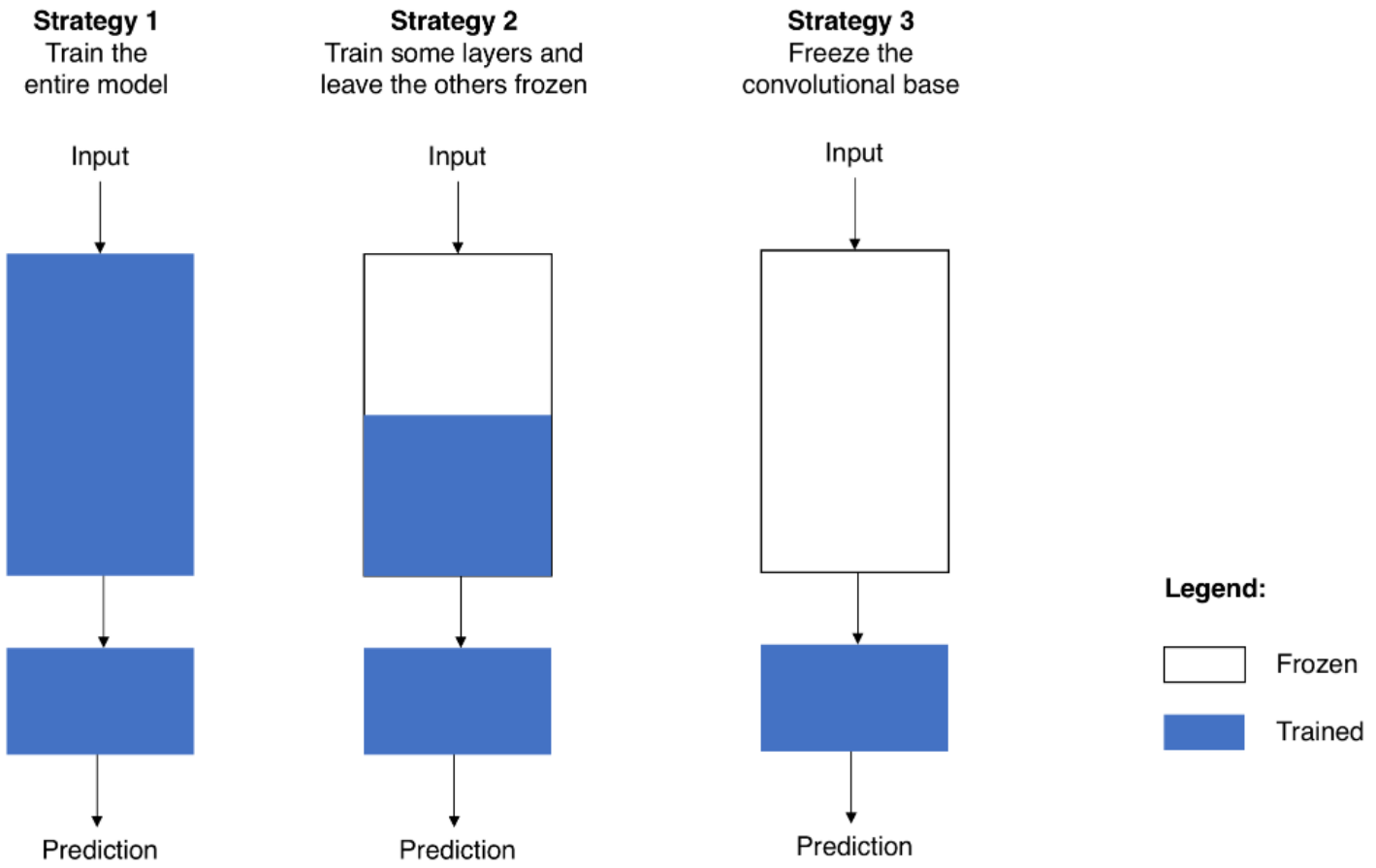
1. **Select Source Task**: You must select a related predictive modeling problem with an abundance of data where there is some relationship in the input data, output data, and/or concepts learned during the mapping from input to output data.
2. **Develop Source Model**: Next, you must develop a skilful model for this first task. The model must be better than a naive model to ensure that some feature learning has been performed.
3. **Reuse Model**: The model fit on the source task can then be used as the starting point for a model on the second task of interest. This may involve using all or parts of the model, depending on the modeling technique used.
4. **Tune Model**.: Optionally, the model may need to be adapted or refined on the input-output pair data available for the task of interest.

Pre-trained Model Approach

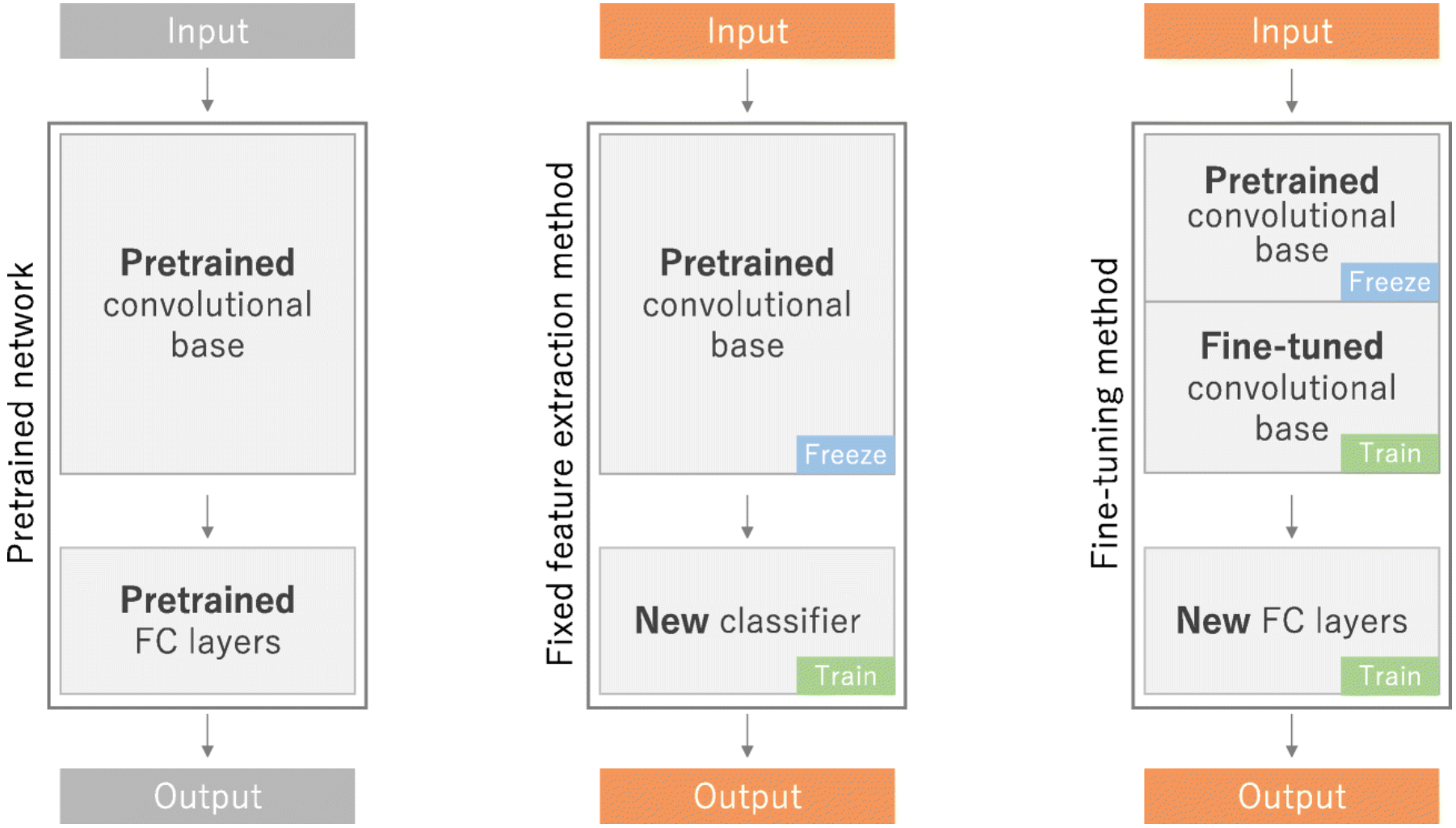
1. **Select Source Model**: A pre-trained source model is chosen from available models. Many research institutions release models on large and challenging datasets that may be included in the pool of candidate models from which to choose from.
2. **Reuse Model**: The model pre-trained model can then be used as the starting point for a model on the second task of interest. This may involve using all or parts of the model, depending on the modeling technique used.
3. **Tune Model**: Optionally, the model may need to be adapted or refined on the input-output pair data available for the task of interest.

- This type of transfer learning is common in the field of deep learning.

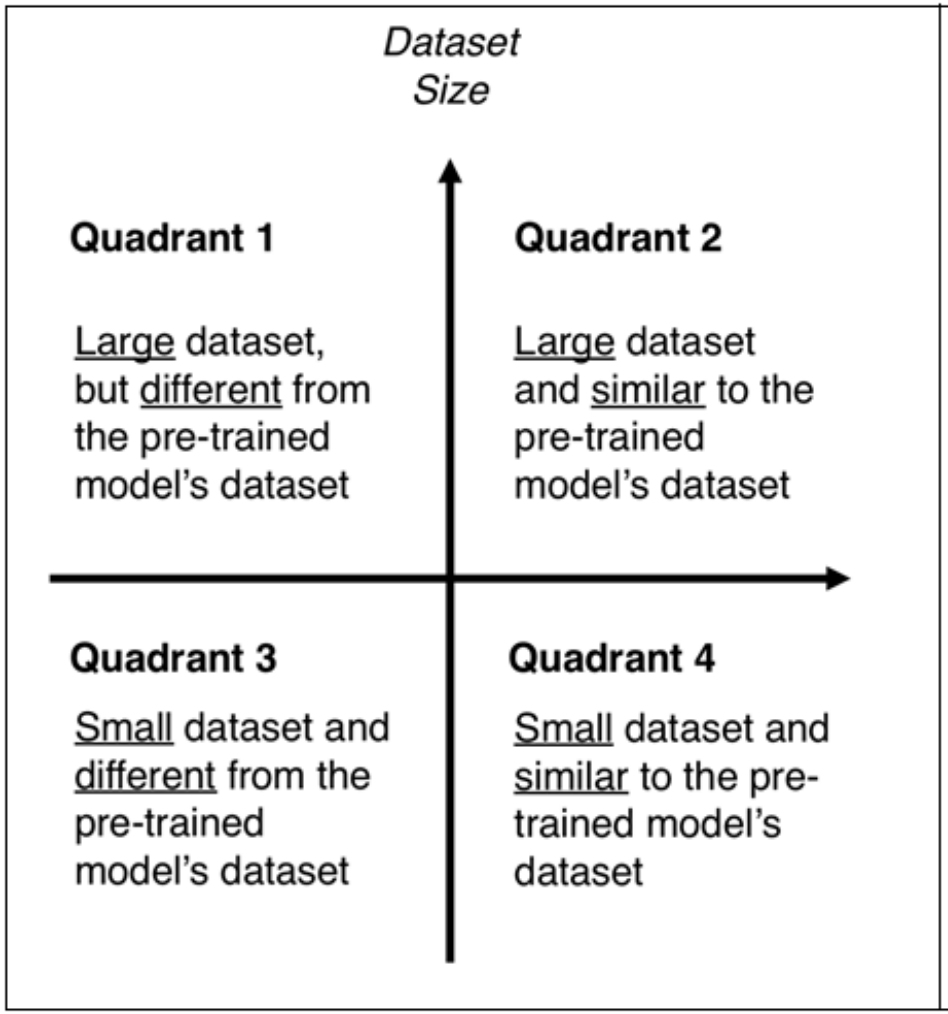
Transfer Learning Strategies

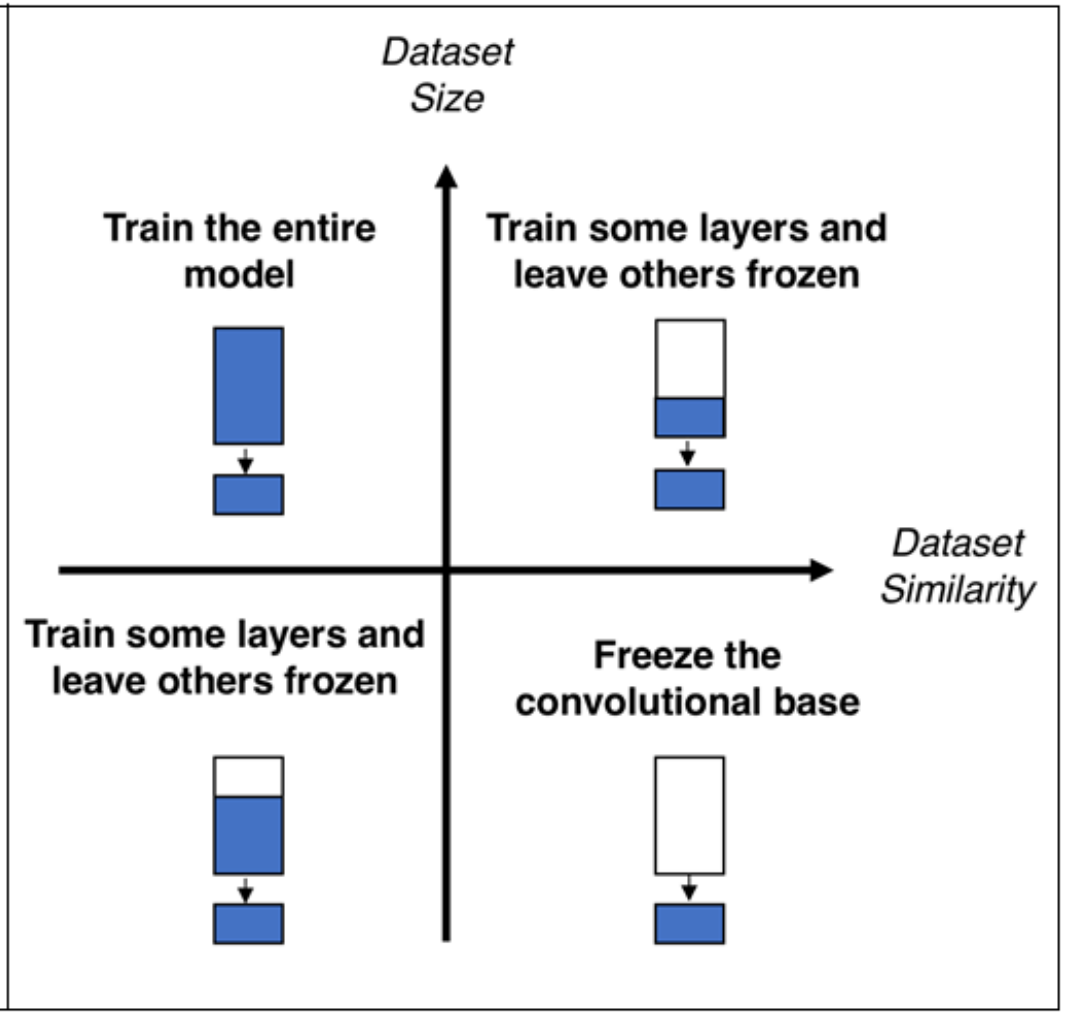


Transfer Learning Strategies



How to select a best strategy..?



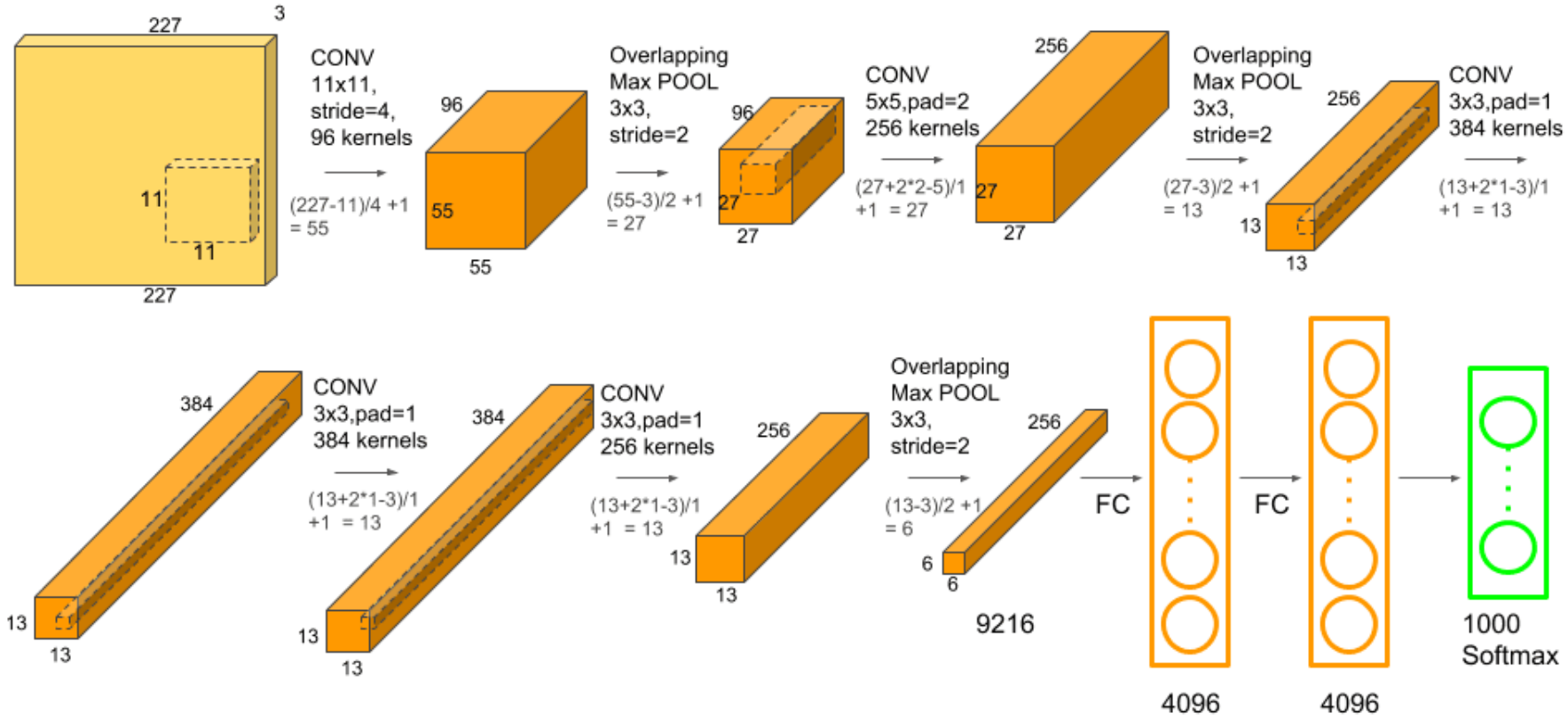


ImageNet Large Scale Visual Recognition Challenge

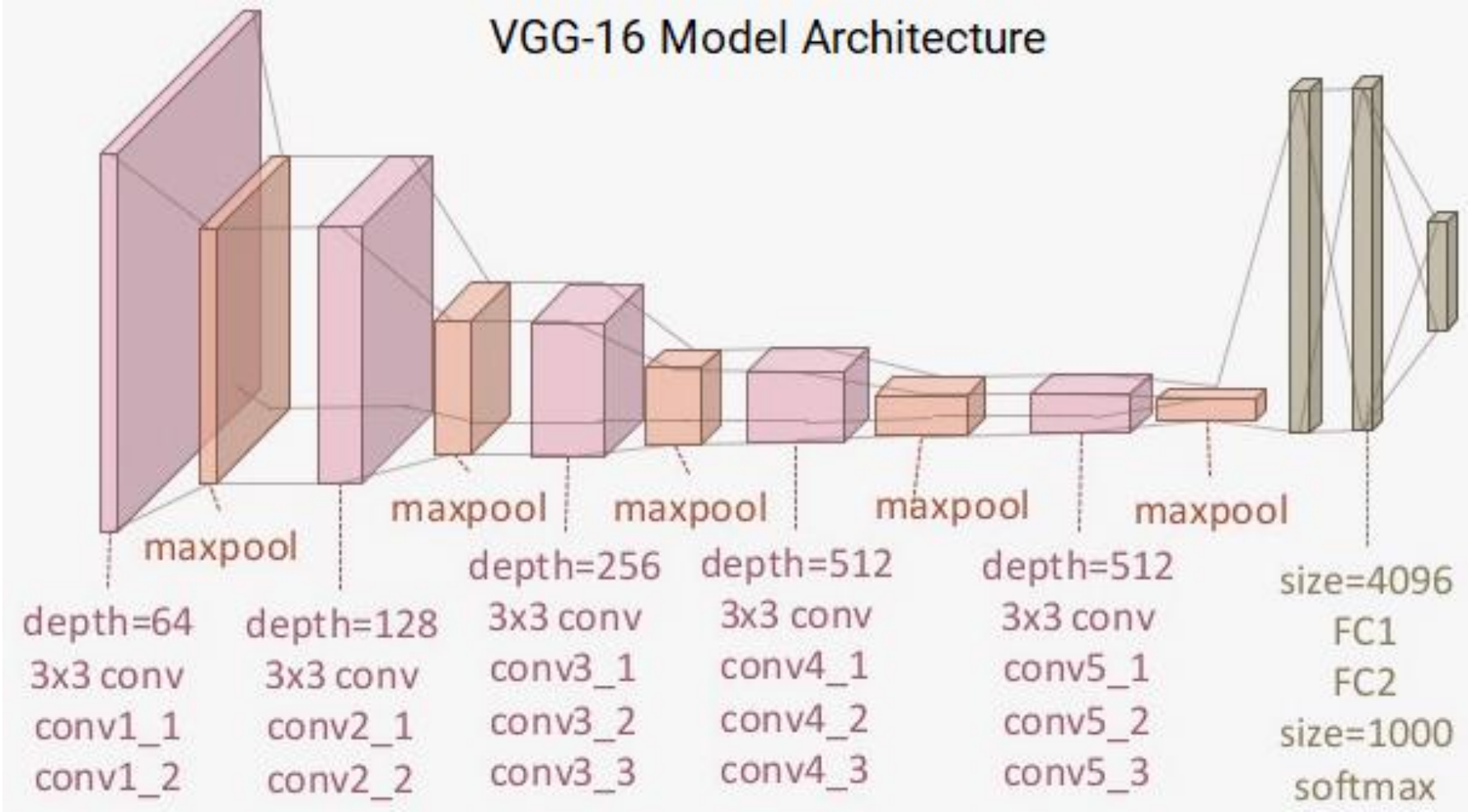
- The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) evaluates algorithms for object detection and image classification at large scale.
- Millions of Labelled Images for Training
- 1000 Classes & Even more now

- To know more: http://www.image-net.org/challenges/LSVRC/

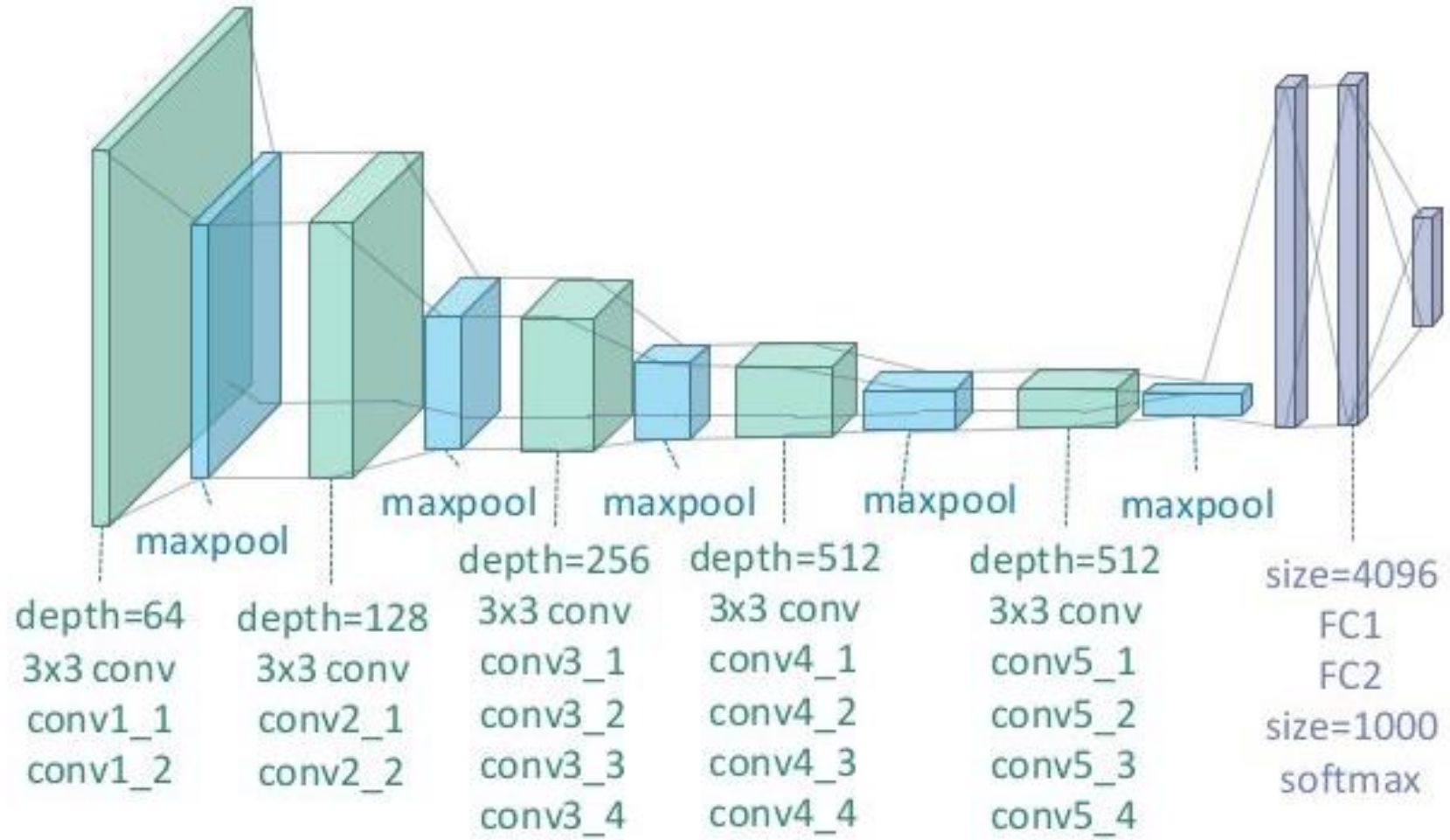
Alex Net


227
3
11
11
227
CONV
11x11,
stride=4,
96 kernels
(227-11)/4 +1
= 55
96
55
55
Overlapping
Max POOL
3x3,
stride=2
(55-3)/2 +1
= 27
96
27
27
CONV
5x5,pad=2
256 kernels
(27+2*2-5)/1
+1 = 27
256
27
27
Overlapping
Max POOL
3x3,
stride=2
(27-3)/2 +1
= 13
256
13
13
CONV
3x3,pad=1
384 kernels
(13+2*1-3)/1
+1 = 13
384
13
13
CONV
3x3,pad=1
384 kernels
(13+2*1-3)/1
+1 = 13
384
13
13
CONV
3x3,pad=1
256 kernels
(13+2*1-3)/1
+1 = 13
256
13
13
Overlapping
Max POOL
3x3,
stride=2
(13-3)/2 +1
= 6
256
6
6
9216
FC
4096
FC
4096
1000
Softmax

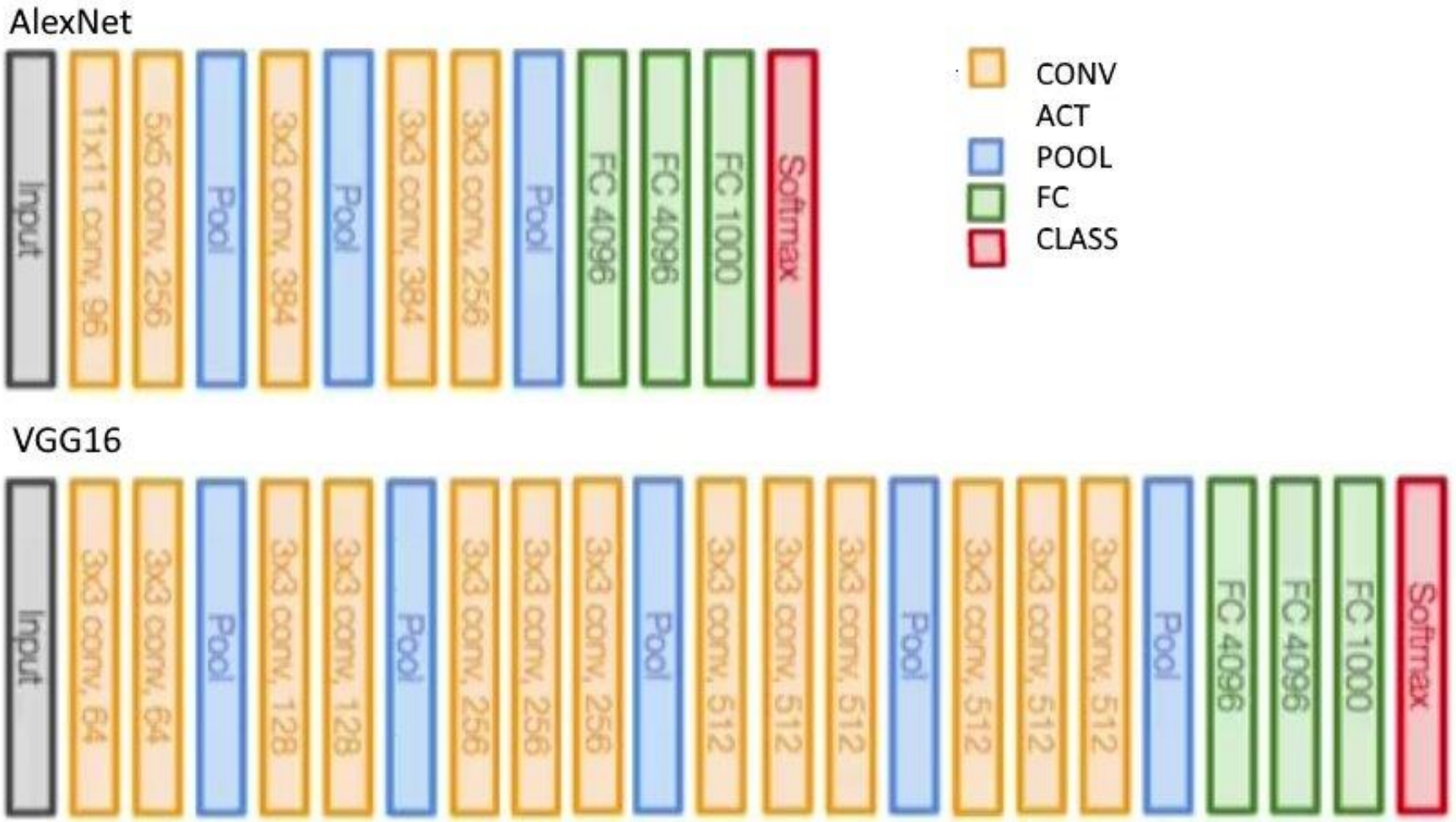
VGG Net (16 Layers – VGG16)



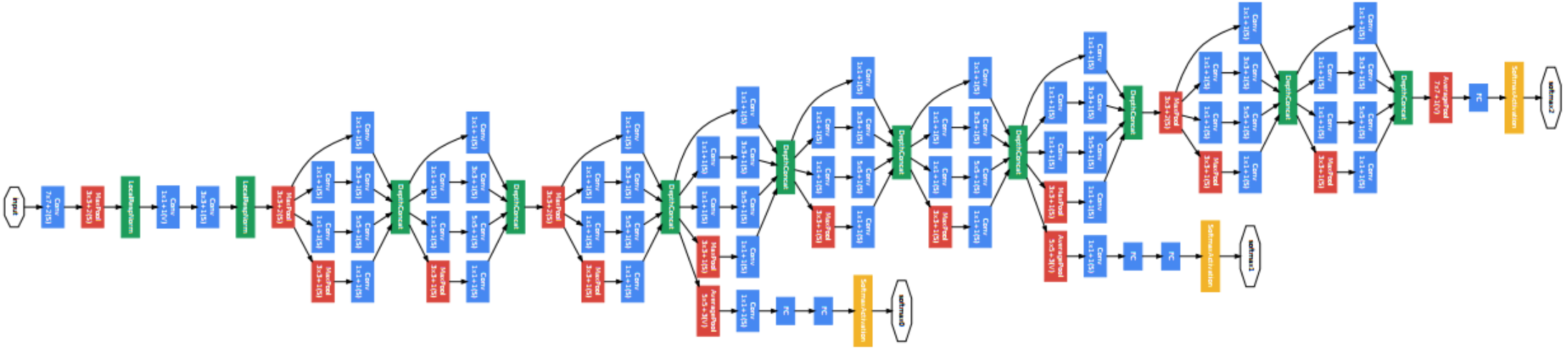
VGG Net (19 Layers – VGG19)

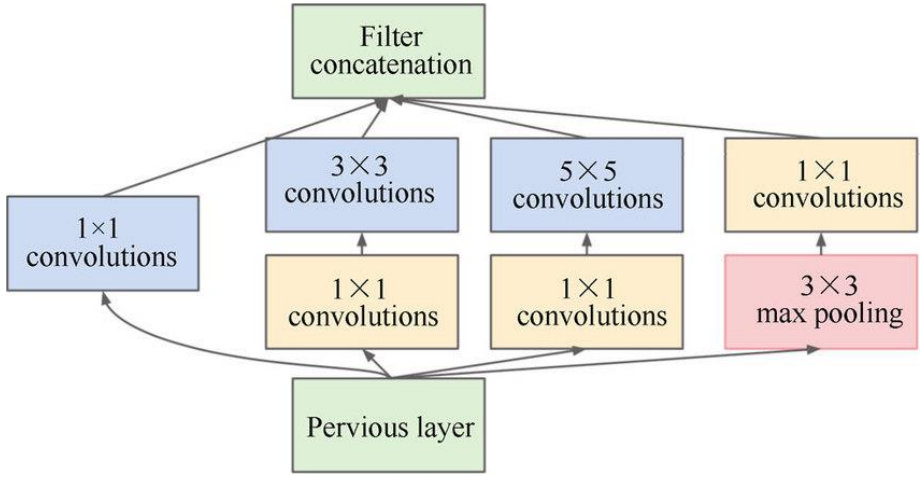


Alex Net vs VGG Net

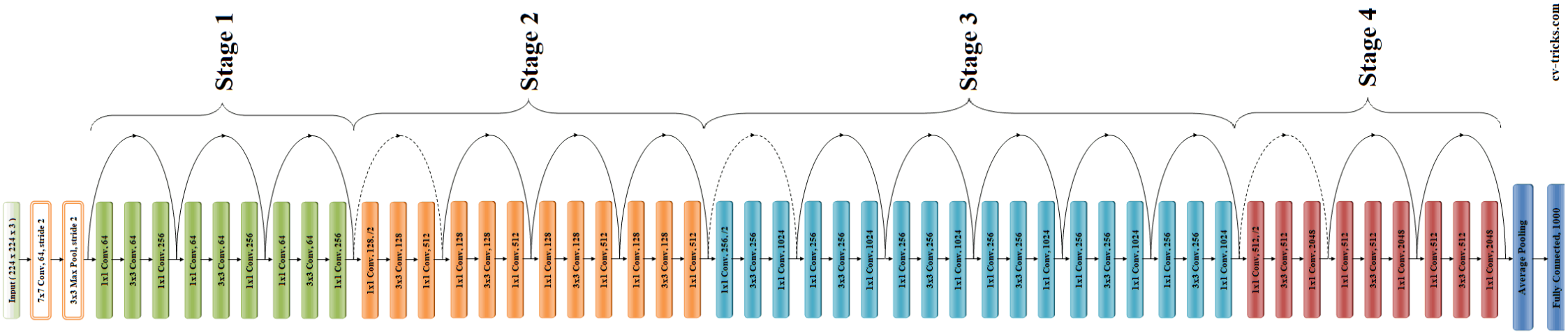


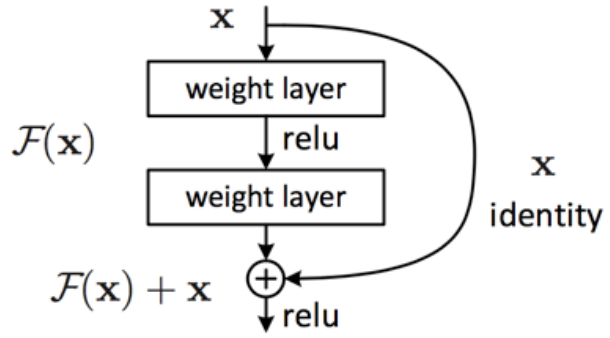
GoogLe Net - Inception





ResNet (50 Layers)





Performance of various Deep CNNs

Model	Top-1 accuracy	Top-5 accuracy
AlexNet	0.625	0.86
VGG-16	0.715	0.901
Inception	0.782	0.941
ResNet-152	0.870	0.963

Other Pre-trained Models

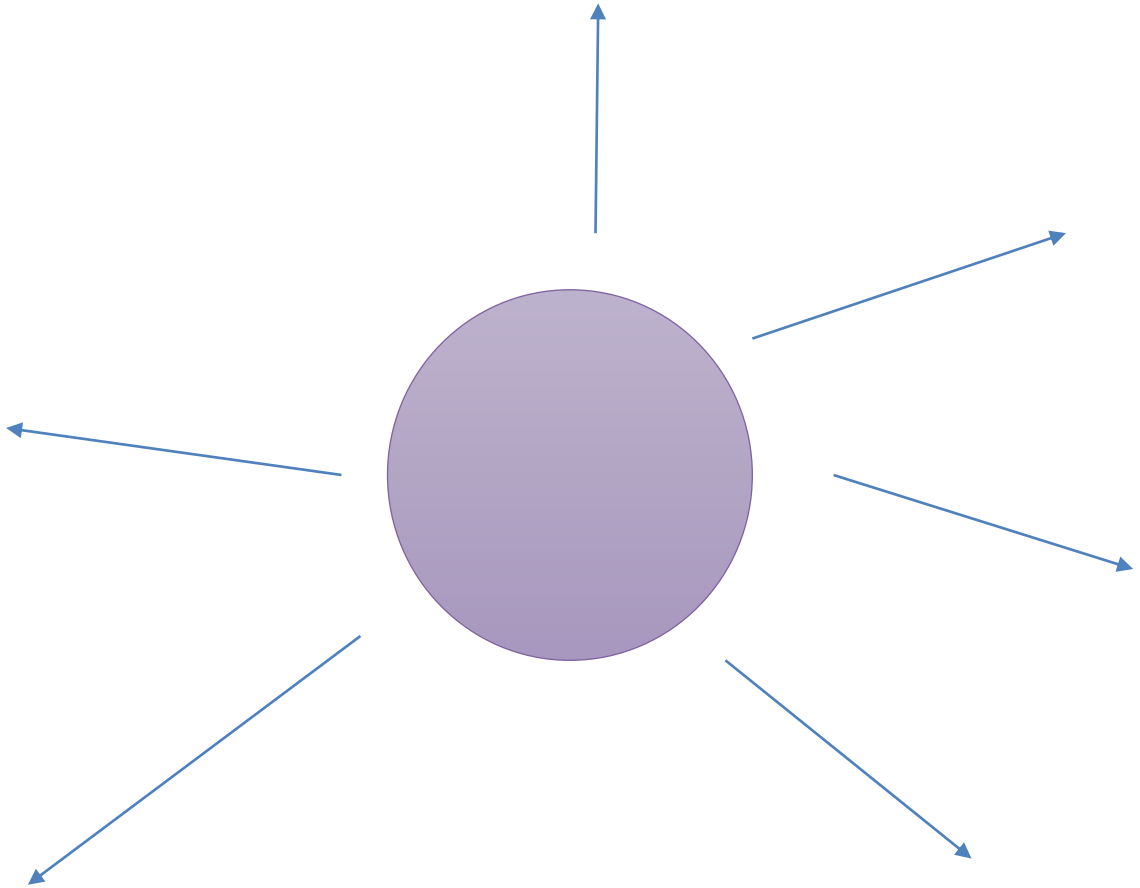
- Xception
- InceptionResNetV2
- MobileNet (V2)
- EfficientNetB0-B7
- NASNet (Mobile & Large)

Using VGG16 Pre-trained Model for

1. Fine-tuning
2. Feature Extraction

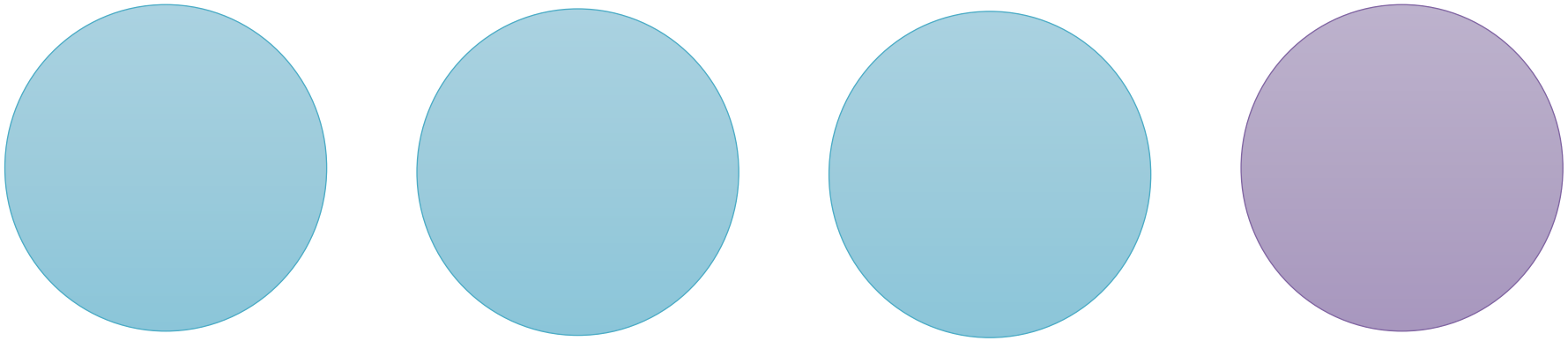
https://keras.io/guides/transfer_learning/

Where this ball would go Next..?



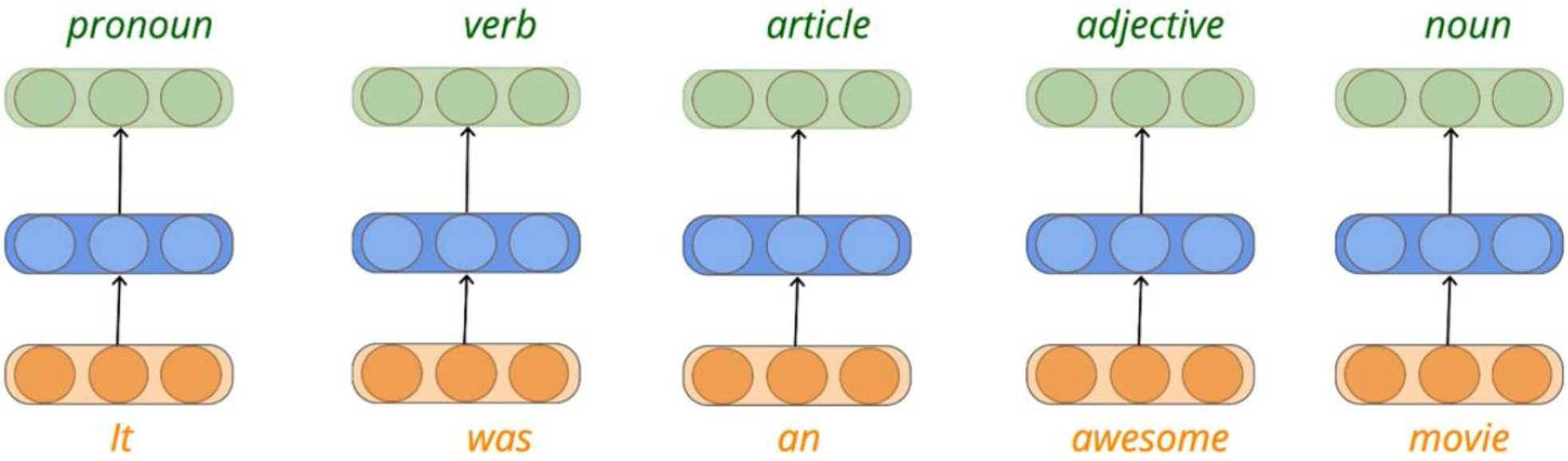
?

Give Previous Positions…

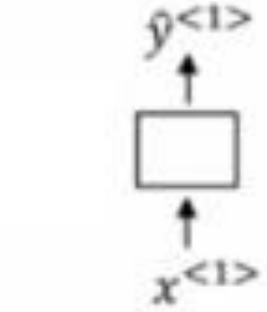


Sequence Modeling

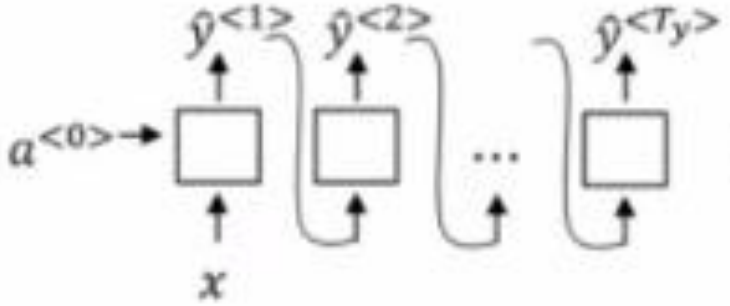
- Sequence Modeling is the task of predicting what word/letter comes next.
- Unlike the DNN and CNN, in sequence modeling, the current output is dependent on the previous input and the length of the input is not fixed.



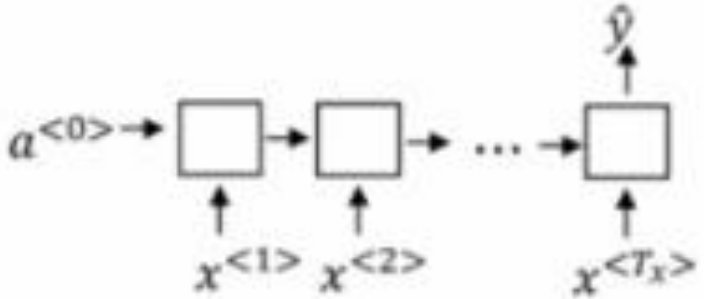
Types of Sequence Modeling Tasks



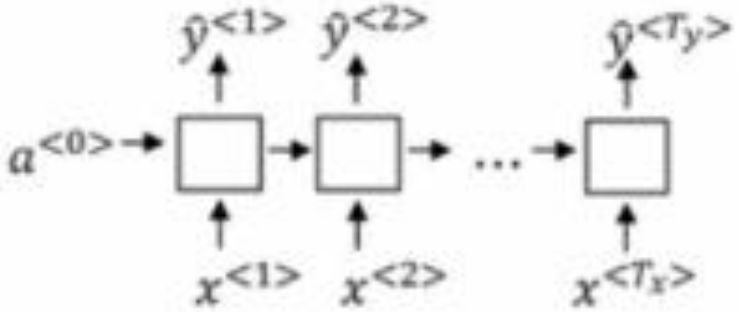
One to one



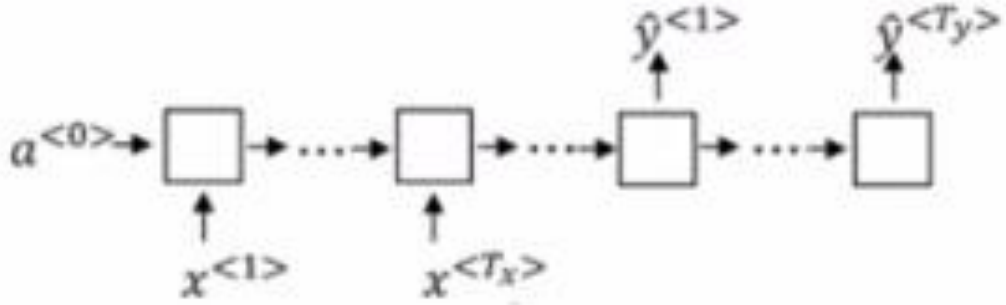
One to many



Many to one



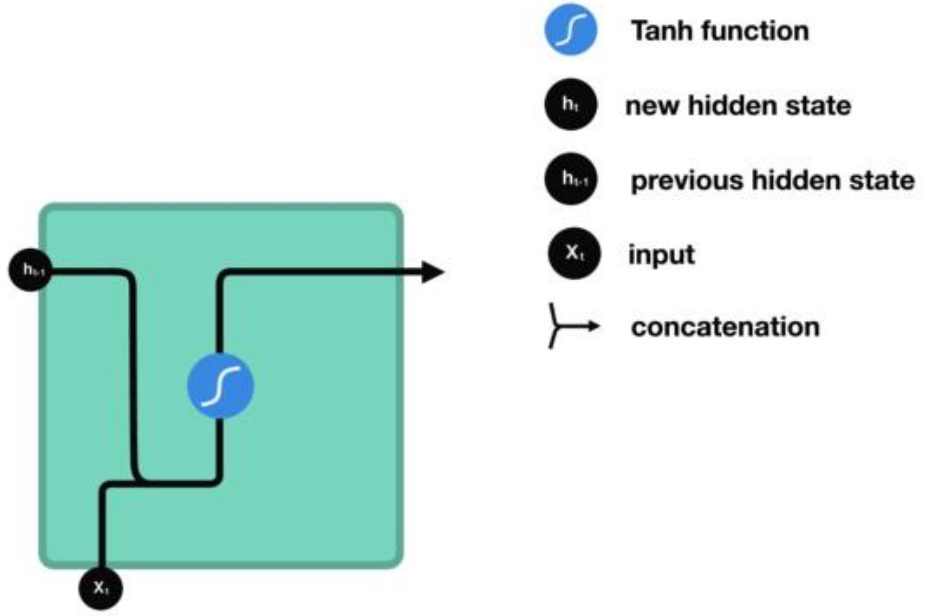
Many to many



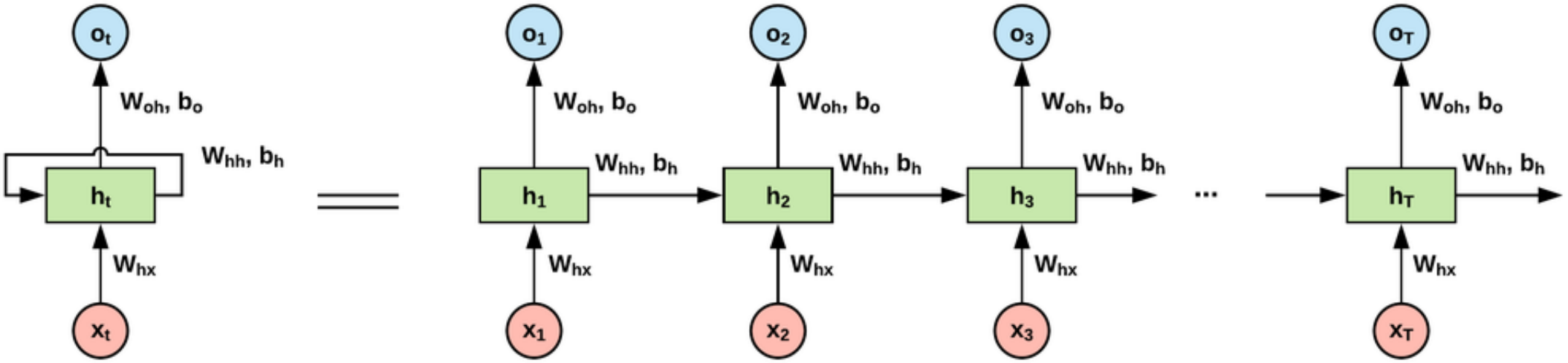
Many to many

Recurrent Neural Network (RNN)

- A recurrent neural network is a class of artificial neural network where connections between nodes form a directed graph along a sequence.
- This allows it to exhibit temporal dynamic behavior for a time sequence.
- Unlike feedforward neural networks, RNNs can use their internal state to process sequences of inputs

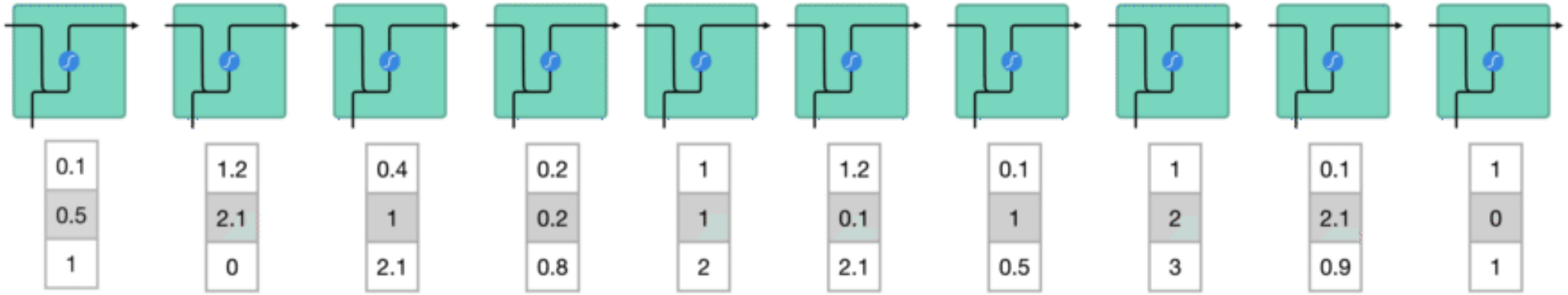


Unfolding RNN

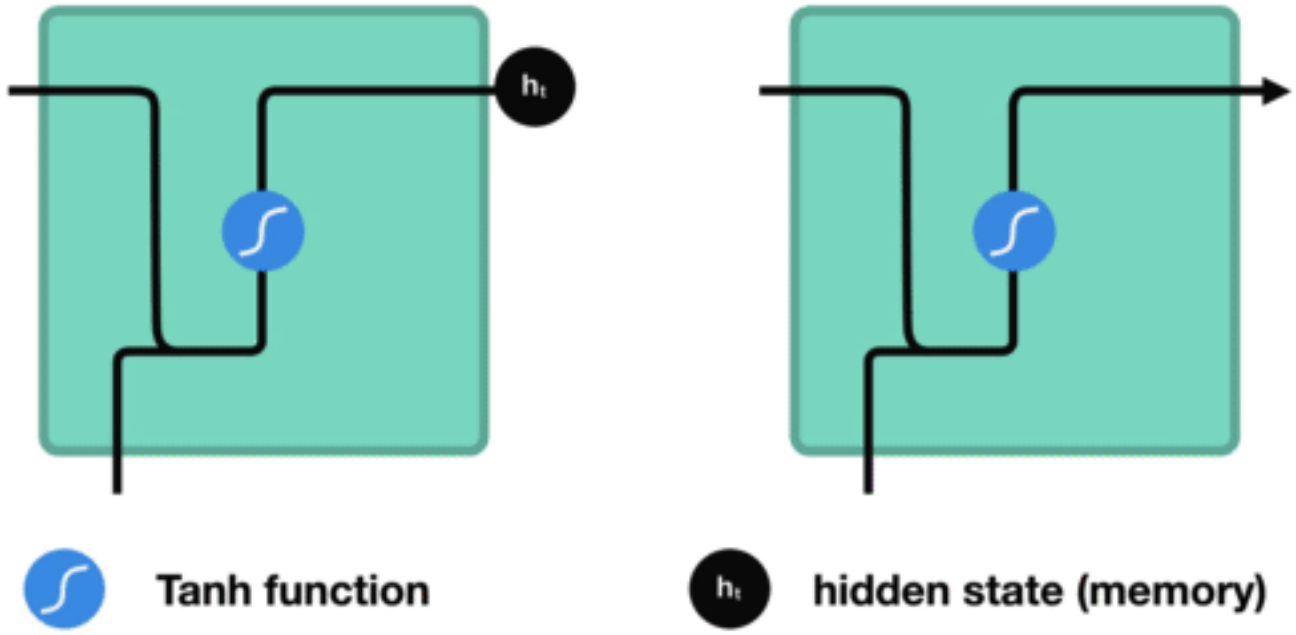


RNN

- An RNN works like this; First words get transformed into machine-readable vectors.
- Then the RNN processes the sequence of vectors one by one.



Hidden State Passing in RNN



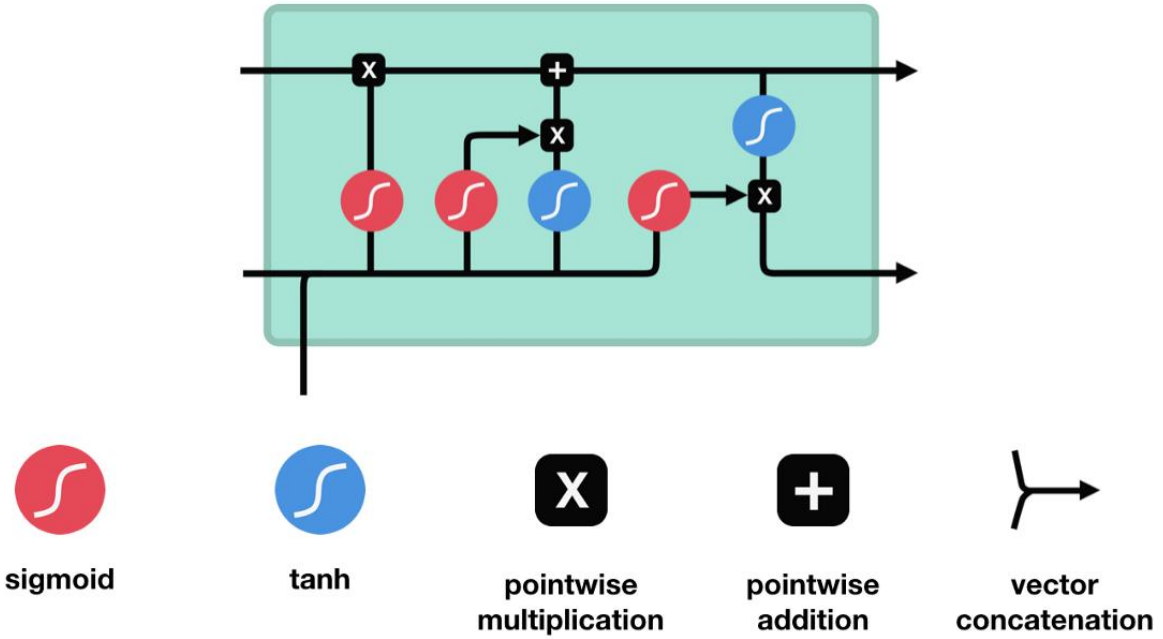
Problem with RNN

- RNNs suffer from the problem of vanishing gradients, which hampers learning of long data sequences.
- The gradients carry information used in the RNN parameter update and when the gradient becomes smaller and smaller, the parameter updates become insignificant which means no real learning is done.

Implementing RNN for Sentiment Analysis

Long Short Time Memory - LSTM

- An LSTM has a similar control flow as a recurrent neural network.
- It processes data passing on information as it propagates forward.
- The differences are the operations within the LSTM's cells.
- These operations are used to allow the LSTM to keep or forget information.

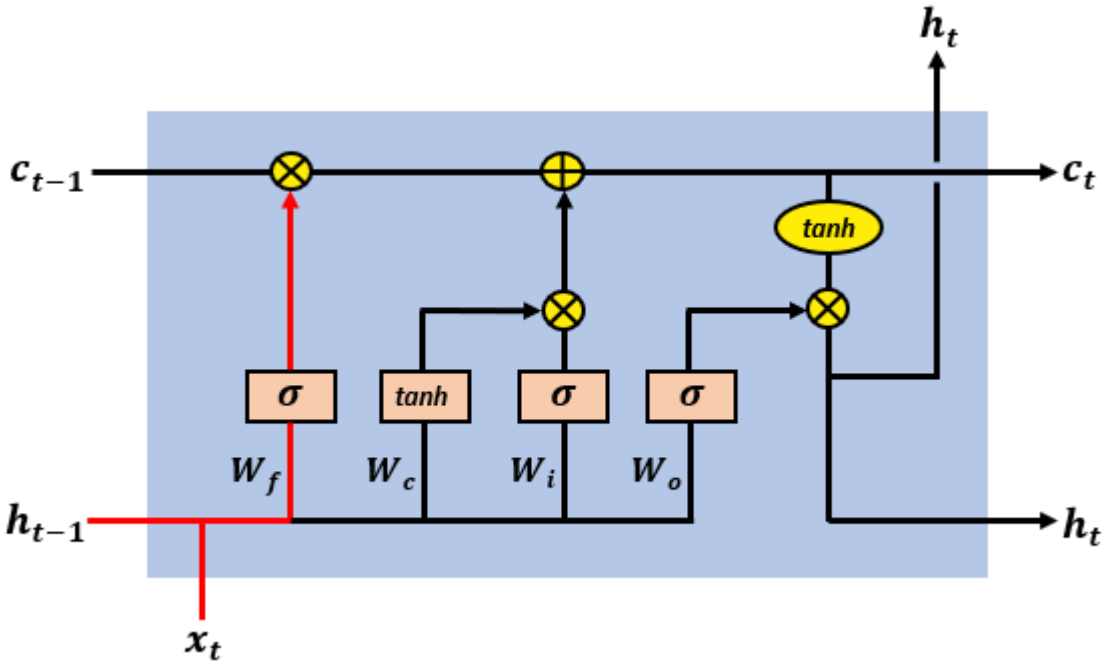


How LSTM Works..?

- The core concept of LSTM's are the cell state, and it's various gates.
- The cell state act as a transport highway that transfers relative information all the way down the sequence chain.
- You can think of it as the "memory" of the network.
- The cell state, in theory, can carry relevant information throughout the processing of the sequence.
- So even information from the earlier time steps can make it's way to later time steps, reducing the effects of short-term memory.
- As the cell state goes on its journey, information get's added or removed to the cell state via gates.
- The gates are different neural networks that decide which information is allowed on the cell state.
- The gates can learn what information is relevant to keep or forget during training.

LSTM: Forget Gate

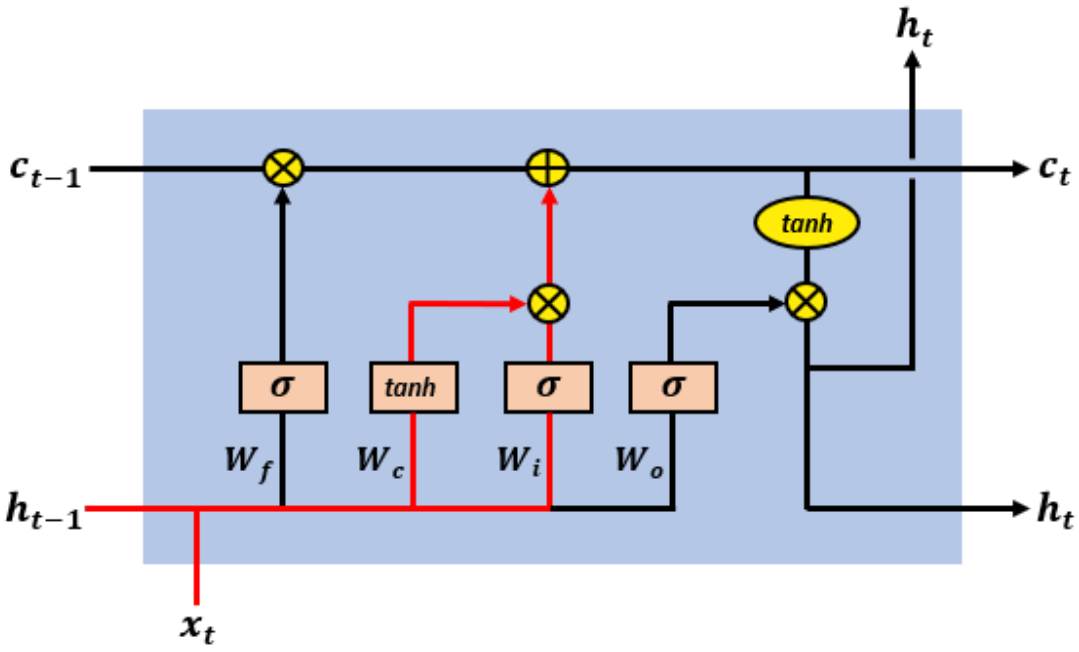
- Forget gate decides what information should be thrown away or kept. Information from the previous hidden state and information from the current input is passed through the sigmoid function. Values come out between 0 and 1. The closer to 0 means to forget, and the closer to 1 means to keep.



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

LSTM: Input Gate

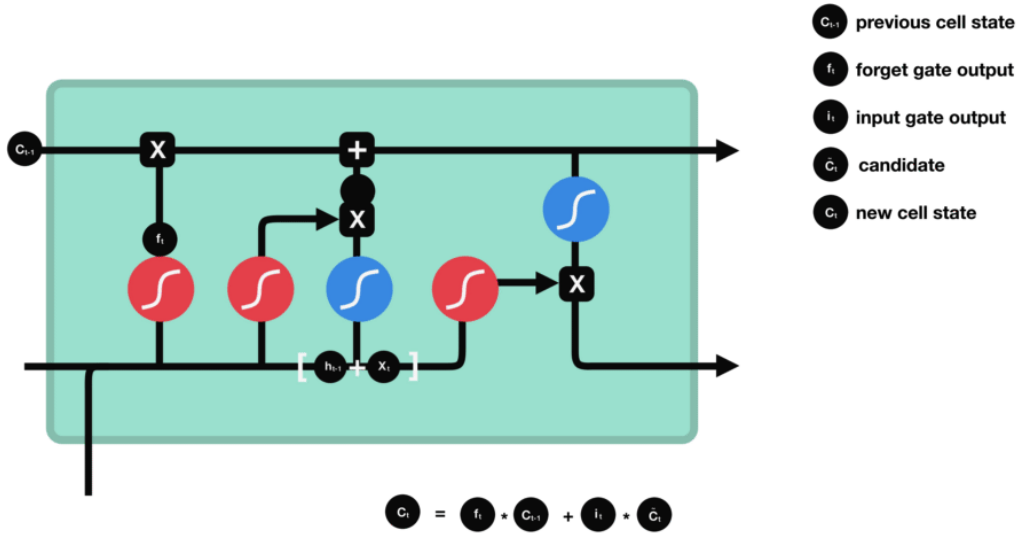
- To update the cell state, we have the input gate.
- First, we pass the previous hidden state and current input into a sigmoid function.
- That decides which values will be updated by transforming the values to be between 0 and 1. 0 means not important, and 1 means important.
- You also pass the hidden state and current input into the tanh function to squish values between -1 and 1 to help regulate the network.
- Then you multiply the tanh output with the sigmoid output.
- The sigmoid output will decide which information is important to keep from the tanh output.



$$tanh(W_c \cdot [h_{t-1}, x_t]) \otimes \sigma(W_i \cdot [h_{t-1}, x_t])$$

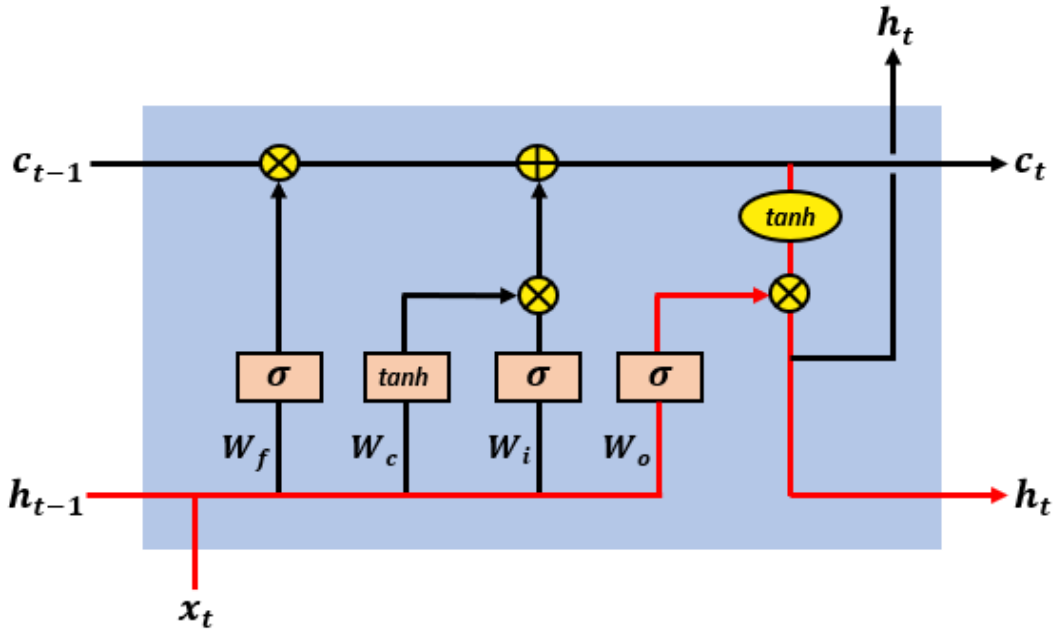
LSTM: Cell State

- First, the cell state gets pointwise multiplied by the forget vector.
- This has a possibility of dropping values in the cell state if it gets multiplied by values near 0.
- Then we take the output from the input gate and do a pointwise addition which updates the cell state to new values that the neural network finds relevant.
- That gives us our new cell state.



LSTM: Output Gate

- The output gate decides what the next hidden state should be.
- Remember that the hidden state contains information on previous inputs. The hidden state is also used for predictions.
- First, we pass the previous hidden state and the current input into a sigmoid function. Then we pass the newly modified cell state to the tanh function.
- We multiply the tanh output with the sigmoid output to decide what information the hidden state should carry. The output is the hidden state. The new cell state and the new hidden is then carried over to the next time step.



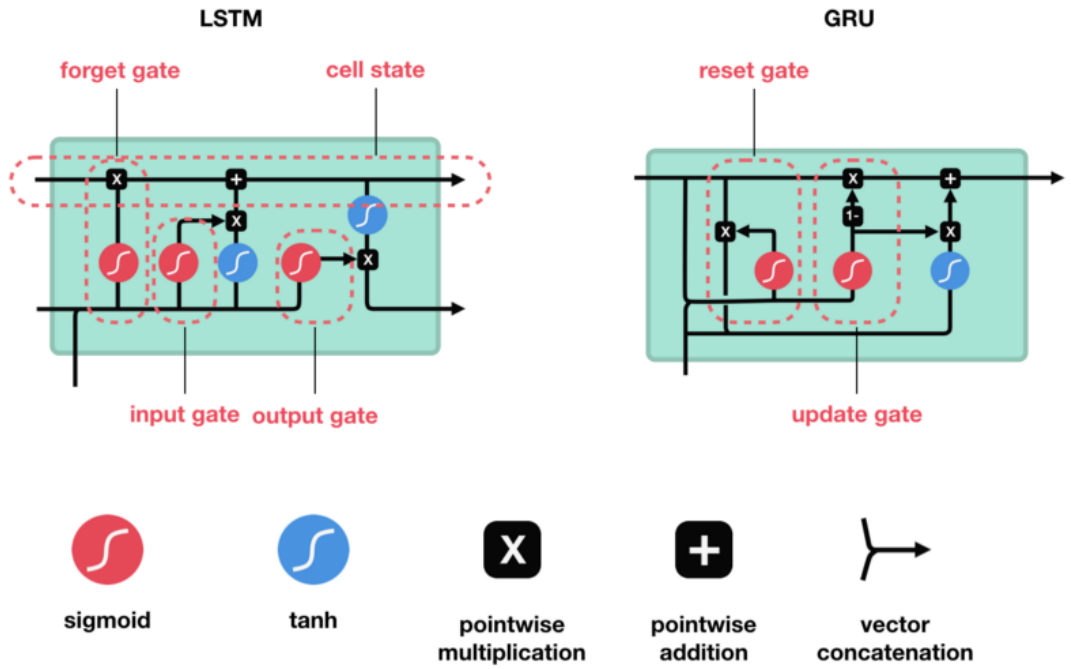
$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t])$$

$$h_t = o_t \otimes tanh(c_t)$$

Implementing LSTM for Sentiment Analysis

Gated Recurrent Unit - GRU

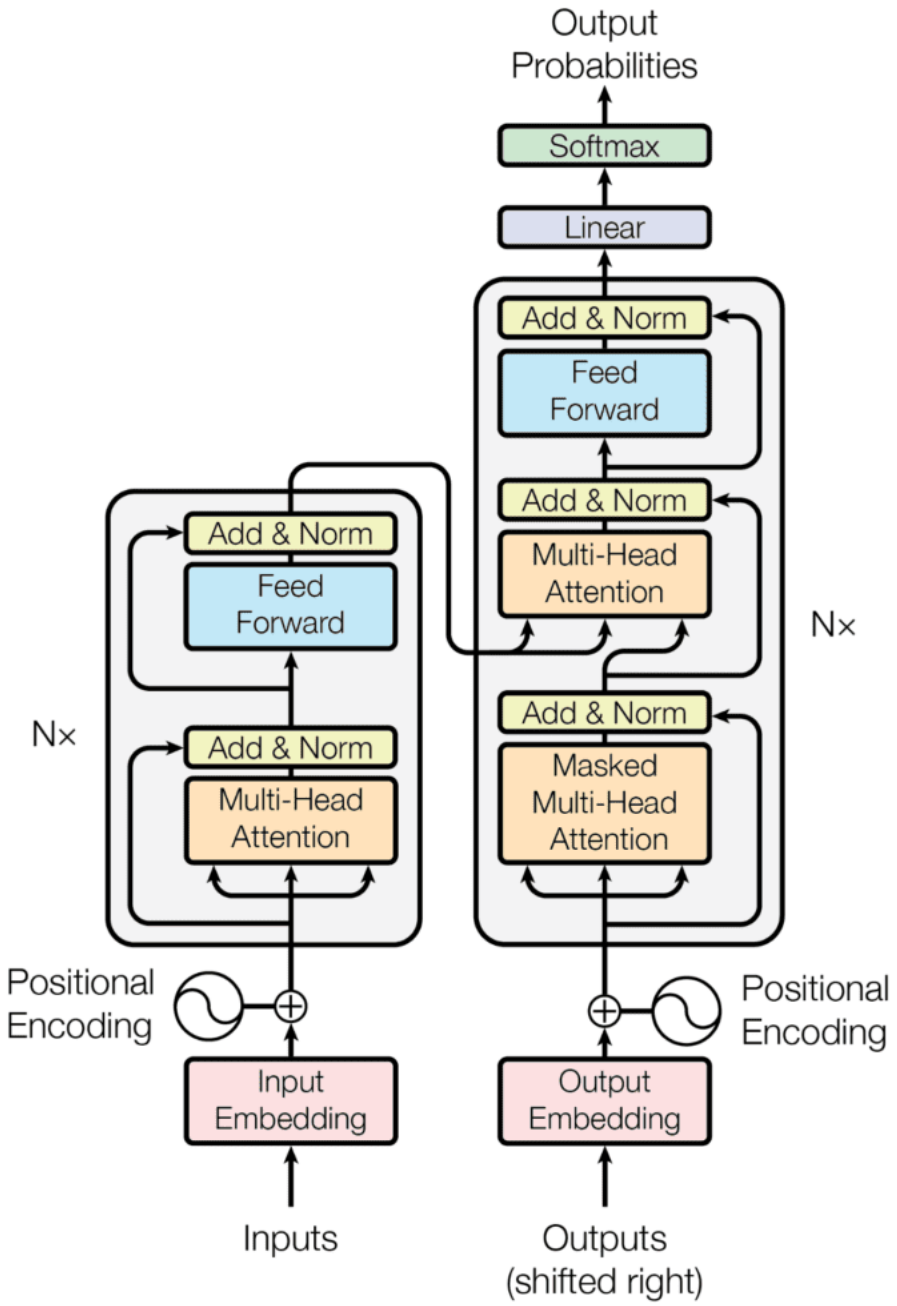
- A slight variation of the LSTM called the Gated recurrent unit(or GRU) was introduced in.
- It uses a single update gate in place of the input and forget gates of the LSTM. It was found to be simpler than the standard LSTM variant and had fewer number of parameters, which resulted in faster training time than LSTMs.



Implementing GRU for Sentiment Analysis

Transformer

- The Transformer architecture follows an encoder-decoder structure, but does not rely on recurrence and convolutions in order to generate an output.
- In a nutshell, the task of the encoder, on the left half of the Transformer architecture, is to map an input sequence to a sequence of continuous representations, which is then fed into a decoder.
- The decoder, on the right half of the architecture, receives the output of the encoder together with the decoder output at the previous time step, to generate an output sequence.



Architectural Overview

Working flow of a Transformer

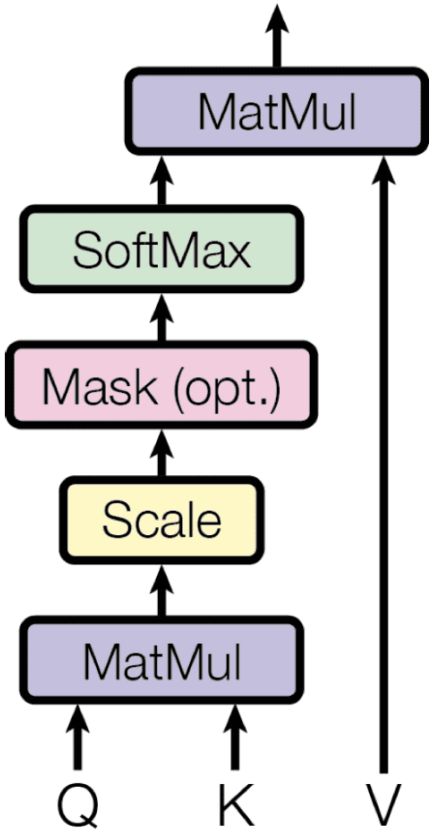
- Each word forming an input sequence is transformed into a d_{model}-dimensional embedding vector.
- Each embedding vector representing an input word is augmented by summing it (element-wise) to a positional encoding vector of the same d_{model} length, hence introducing positional information into the input.
- The augmented embedding vectors are fed into the encoder block, consisting of the two sublayers.
    1. The first sublayer implements a multi-head self-attention mechanism. Multi-head mechanism implements h heads that receive a (different) linearly projected version of the queries, keys and values each, to produce h outputs in parallel that are then used to generate a final result.
    2. The second sublayer is a fully connected feed-forward network, consisting of two linear transformations with Rectified Linear Unit (ReLU) activation in between.
- Since the encoder attends to all words in the input sequence, irrespective if they precede or succeed the word under consideration, then the Transformer encoder is *bidirectional*.

Working flow of a Transformer

- The decoder receives as input its own predicted output word at time-step, t–1.
- The input to the decoder is also augmented by positional encoding, in the same manner as this is done on the encoder side.
- The augmented decoder input is fed into the three sublayers comprising the decoder block.

Working flow of a Transformer

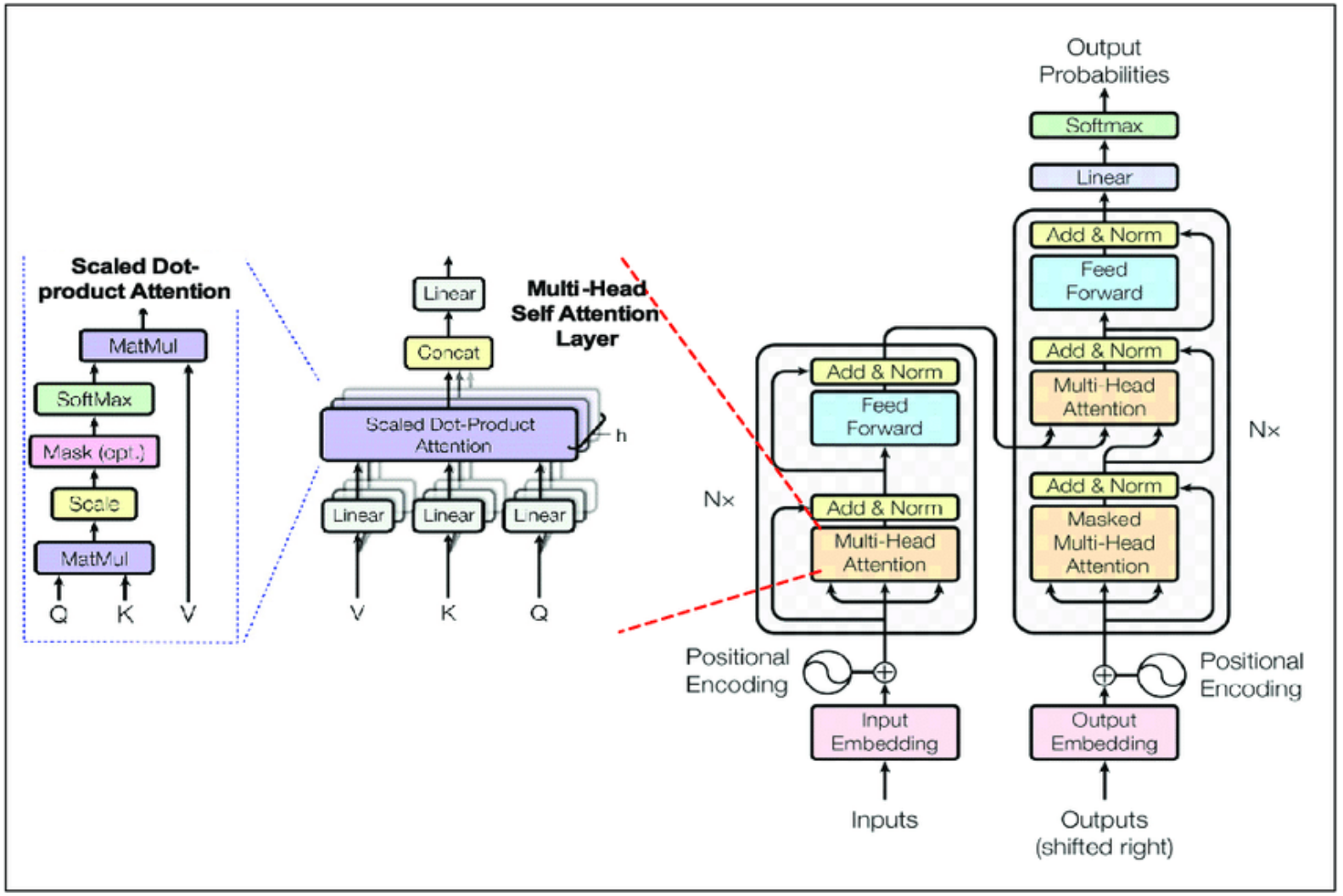
- The first sublayer receives the previous output of the decoder stack, augments it with positional information, and implements multi-head self-attention over it. While the encoder is designed to attend to all words in the input sequence, *regardless* of their position in the sequence, the decoder is modified to attend *only* to the preceding words. Hence, the prediction for a word at position, i, can only depend on the known outputs for the words that come before it in the sequence. In the multi-head attention mechanism (which implements multiple, single attention functions in parallel), this is achieved by introducing a mask over the values produced by the scaled multiplication of matrices Q and K. This masking is implemented by suppressing the matrix values that would, otherwise, correspond to illegal connections.



Working flow of a Transformer

  - The second layer implements a multi-head self-attention mechanism, which is similar to the one implemented in the first sublayer of the encoder. On the decoder side, this multi-head mechanism receives the queries from the previous decoder sublayer, and the keys and values from the output of the encoder. This allows the decoder to attend to all of the words in the input sequence.
  - The third layer implements a fully connected feed-forward network, which is similar to the one implemented in the second sublayer of the encoder.
- Masking is applied in the first sublayer, in order to stop the decoder from attending to succeeding words. At the second sublayer, the decoder also receives the output of the encoder, which now allows the decoder to attend to all of the words in the input sequence.
- The output of the decoder finally passes through a fully connected layer, followed by a softmax layer, to generate a prediction for the next word of the output sequence.

Unfolded Transformer Architecture



Implementing Transformer for Sentiment Analysis

Hands-on Exercises

- ✓ MP Neuron for Diabetes Prediction
- ✓ Perceptron for Diabetes Prediction (scratch & sklearn)
- ✓ MLP for Loan Prediction (sklearn)
- ✓ DNN for Loan Prediction (BP scratch)
- ✓ DNN for Image Classification (tensor flow)
- ✓ CNN for Image Classification (tensor flow)
- ✓ Regularized CNN for Image Classification (tensor flow)
- ✓ VGG16 for Image Classification (Fine-tuning Approach)
- ✓ VGG16 for Image Classification (Feature Extraction Approach)
- ✓ RNN for Sentiment Analysis
- ✓ LSTM for Sentiment Analysis
- ✓ GRU for Sentiment Analysis
- ✓ Transformer for Sentiment Analysis