Deep Learning

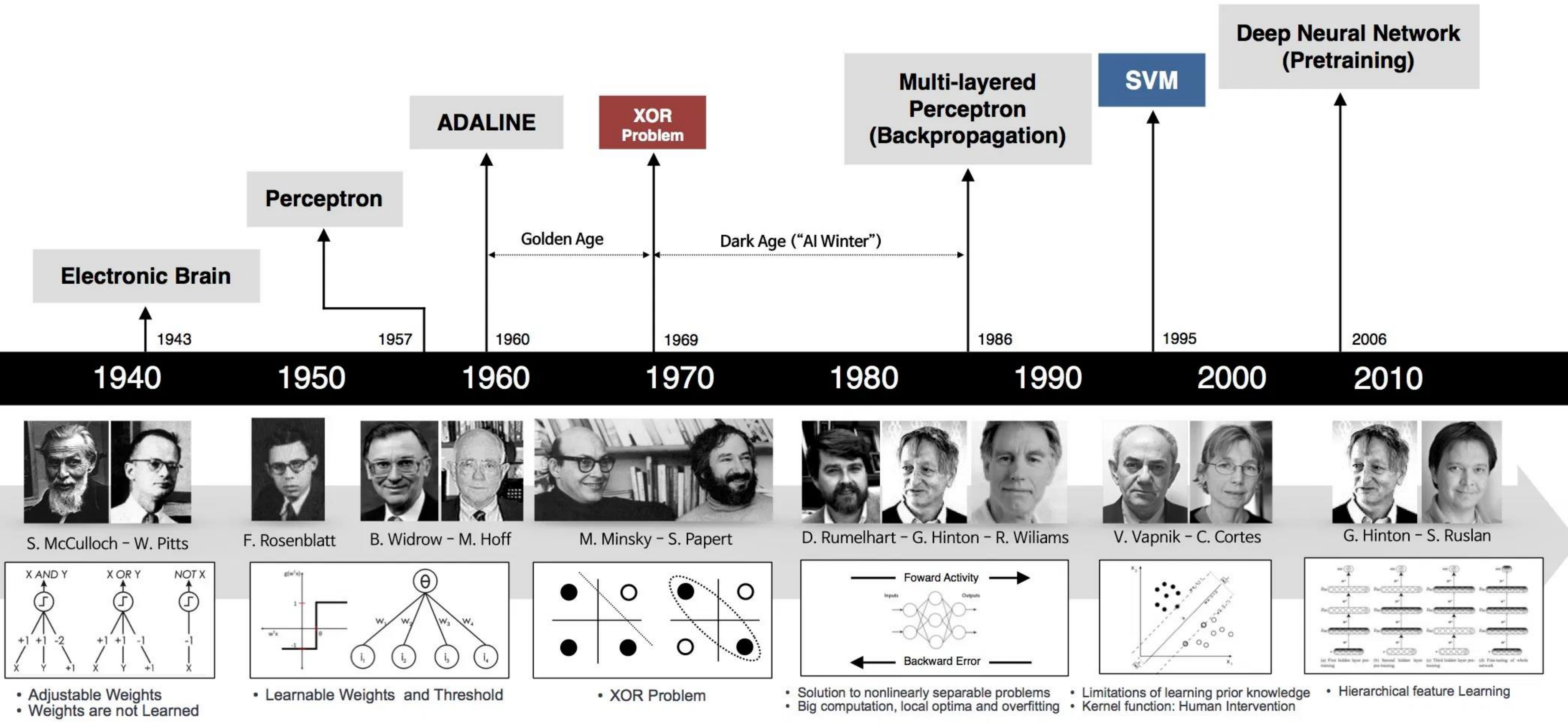
18 August 2020

https://www.linkedin.com/in/arunkumarnair/

Mobile: +91-9890652675

Email: arunkg99@gmail.com

History



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

Machine Learning and Deep Learning

What is Learning

- A machine-learning system is *trained* rather than explicitly programmed. It’s presented with many examples relevant to a task, and it **finds statistical structure** in these examples that eventually allows the system to come up with rules for automating the task.
- For instance, if you wished to automate the task of tagging your vacation pictures, you

What is Leanring

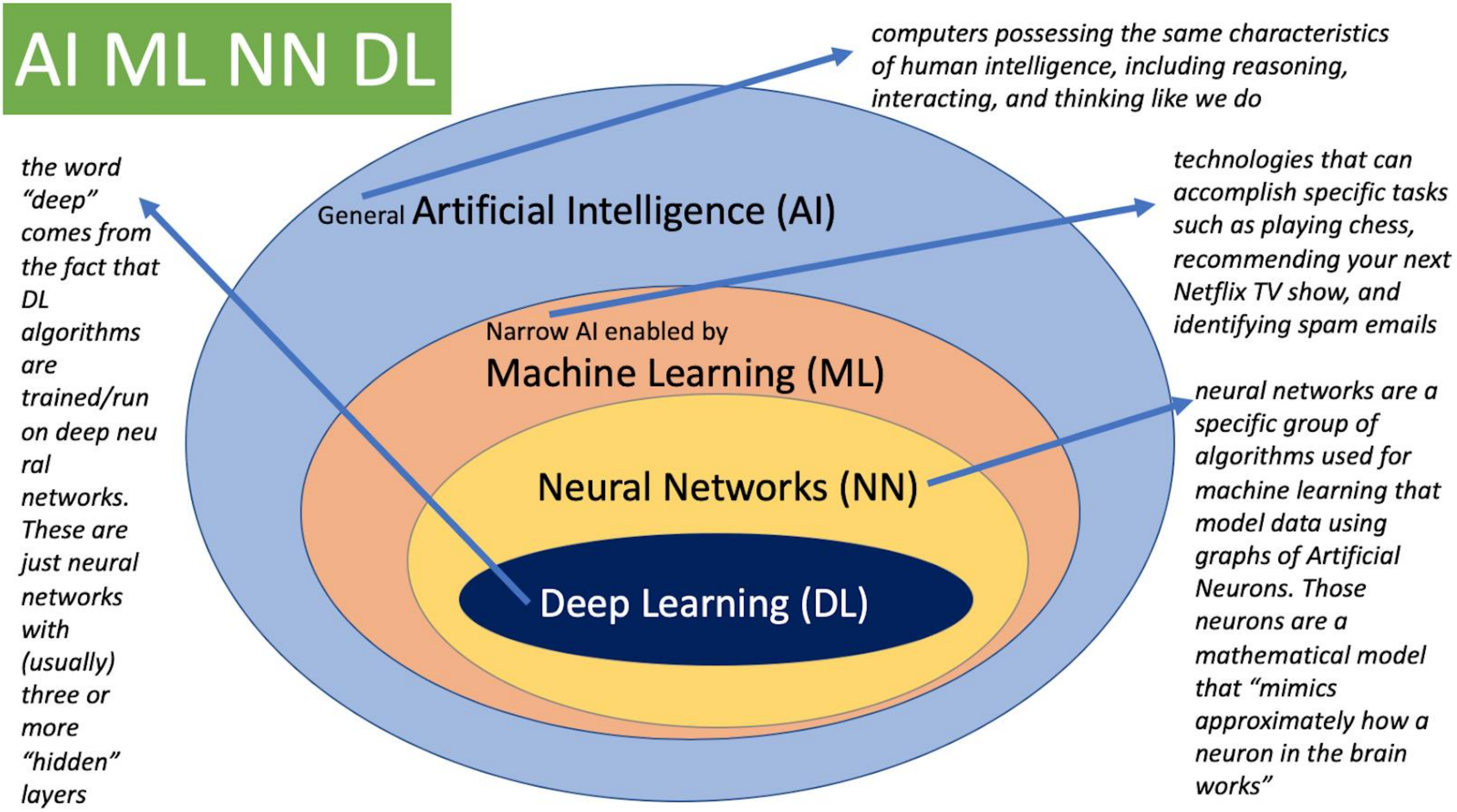
could present a machine-learning system with many examples of pictures already tagged by humans, and the system would learn statistical rules for associating specific pictures to specific tags.

Learned

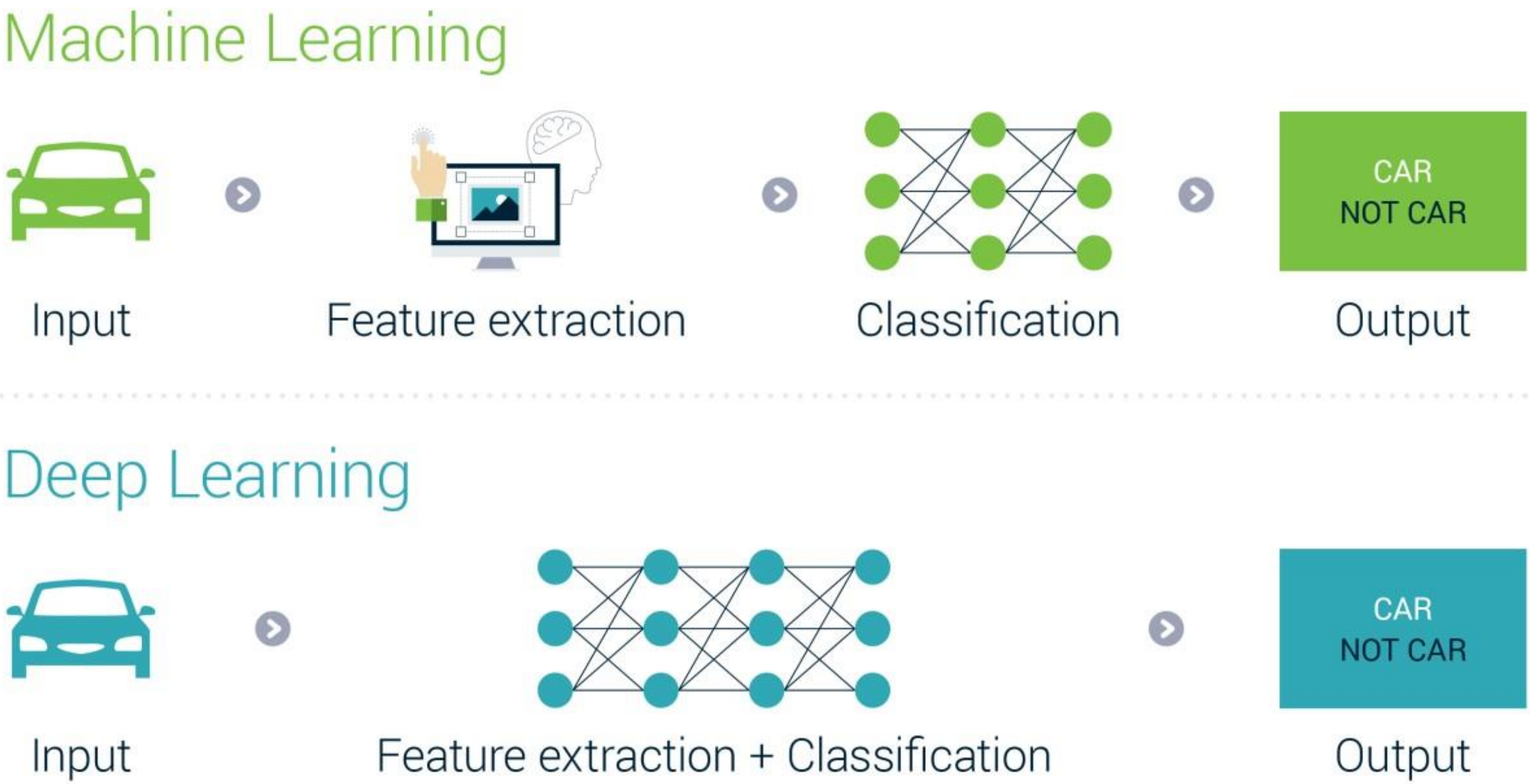
- A machine-learning model transforms its input data into meaningful outputs, a process that is "learned" from exposure to known examples of inputs and outputs. Therefore, the central problem in machine learning and deep learning is to *meaningfully transform data*: in other words, to learn useful *representations* of the input data at hand—representations that get us closer to the expected output.

Deeplearning

- How many layers contribute to a model of the data is called the *depth* of the model. Other appropriate names for the field could have been *layered representations learning* and *hierarchical representations learning*.


AI ML NN DL
computers possessing the same characteristics of human intelligence, including reasoning, interacting, and thinking like we do
General Artificial Intelligence (AI)
technologies that can accomplish specific tasks such as playing chess, recommending your next Netflix TV show, and identifying spam emails
Narrow AI enabled by
Machine Learning (ML)
neural networks are a specific group of algorithms used for machine learning that model data using graphs of Artificial Neurons. Those neurons are a mathematical model that “mimics approximately how a neuron in the brain works”
Neural Networks (NN)
Deep Learning (DL)
the word “deep” comes from the fact that DL algorithms are trained/run on deep neu ral networks. These are just neural networks with (usually) three or more “hidden” layers

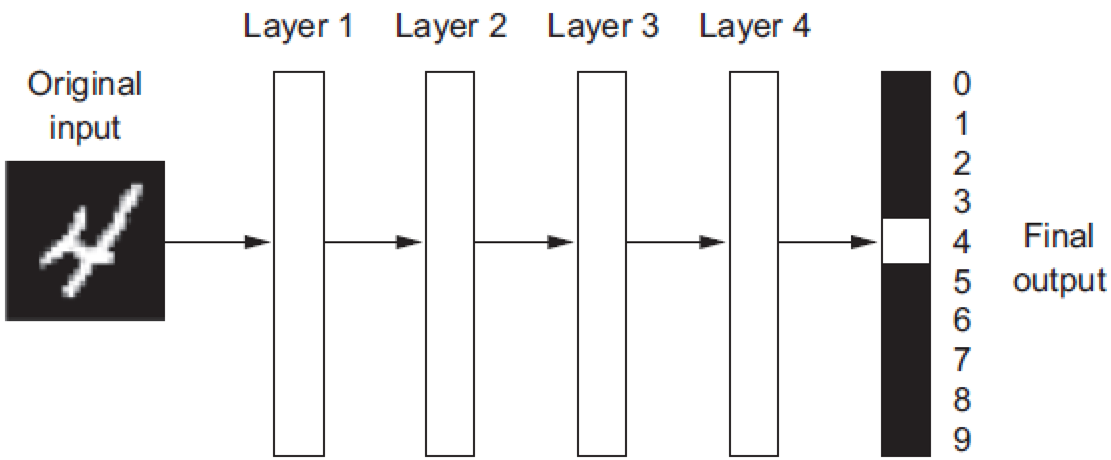
ML vs DL


Machine Learning
CAR
NOT CAR
Input
Feature extraction
Classification
Output
Deep Learning
CAR
NOT CAR
Input
Feature extraction + Classification
Output

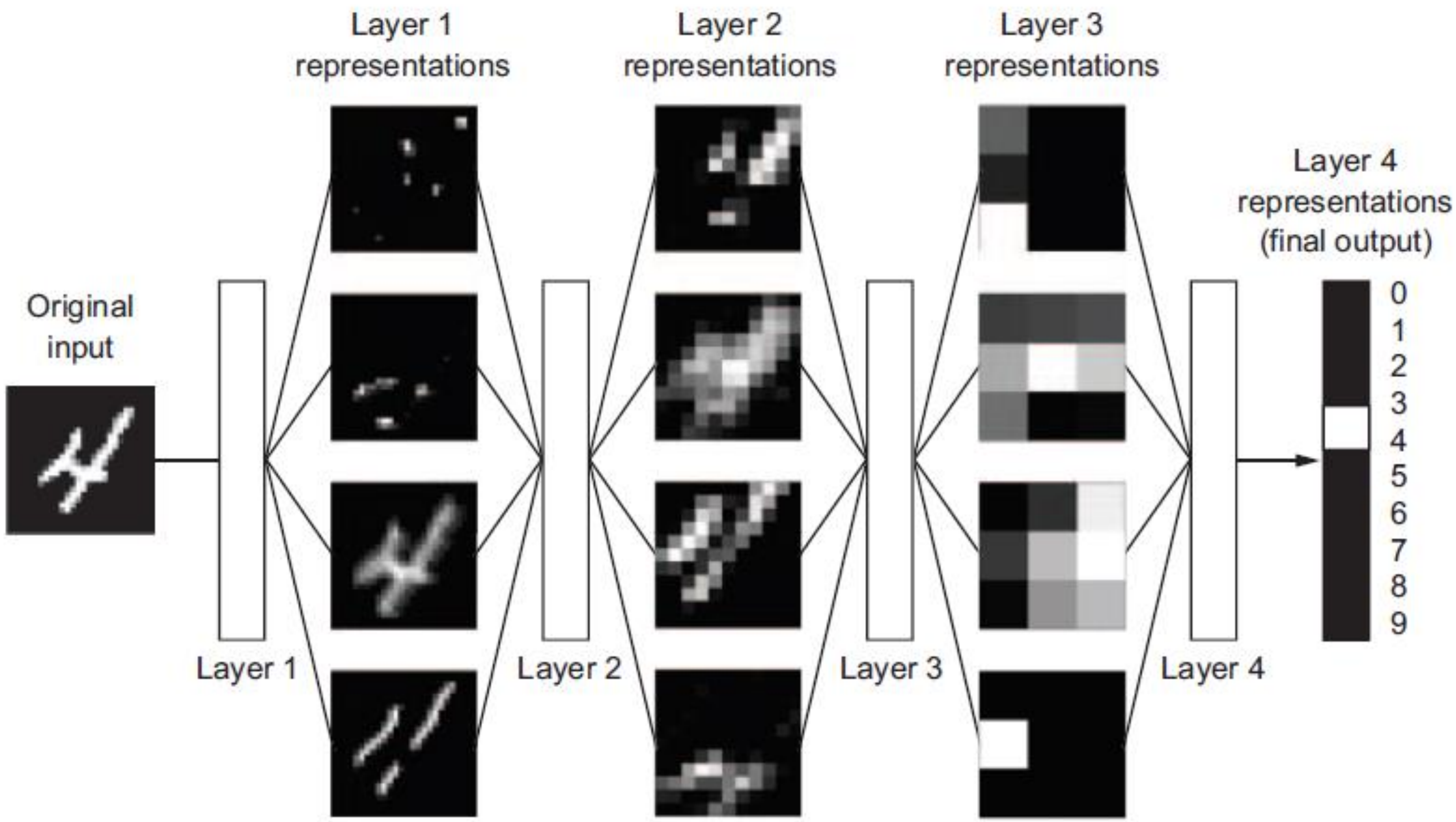
ML vs DL

Factors	Deep Learning	Machine Learning
Data Requirement	Requires large data	Can train on lesser data
Accuracy	Provides high accuracy	Gives lesser accuracy
Training Time	Takes longer to train	Takes less time to train
Hardware Dependency	Requires GPU to train properly	Trains on CPU
Hyperparameter Tuning	Can be tuned in various different ways.	Limited tuning capabilities

What do Representations Learn



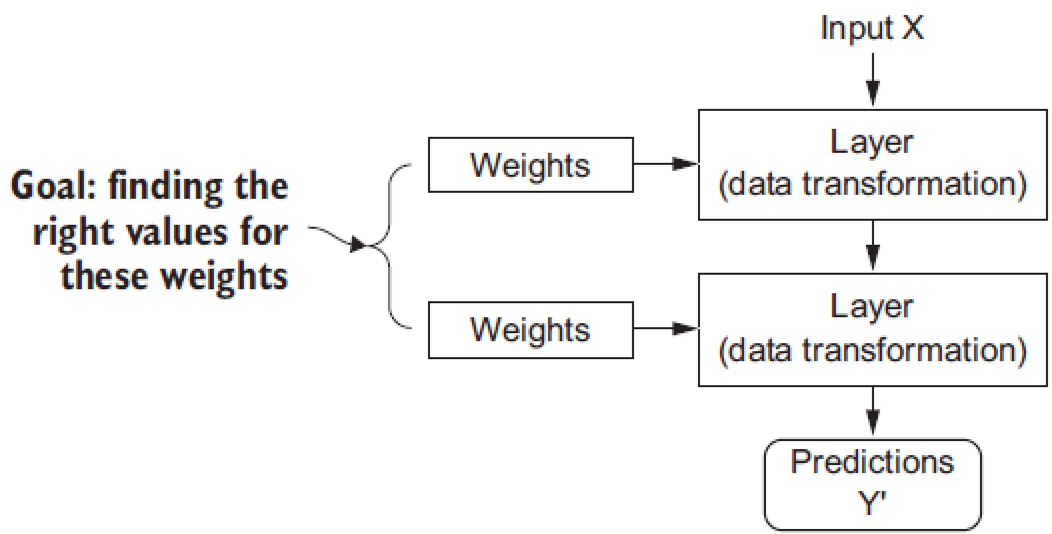
Digit classification



Input Weights

- The specification of what a layer does to its input data is stored in the layer’s *weights*, which in essence are a bunch of numbers.

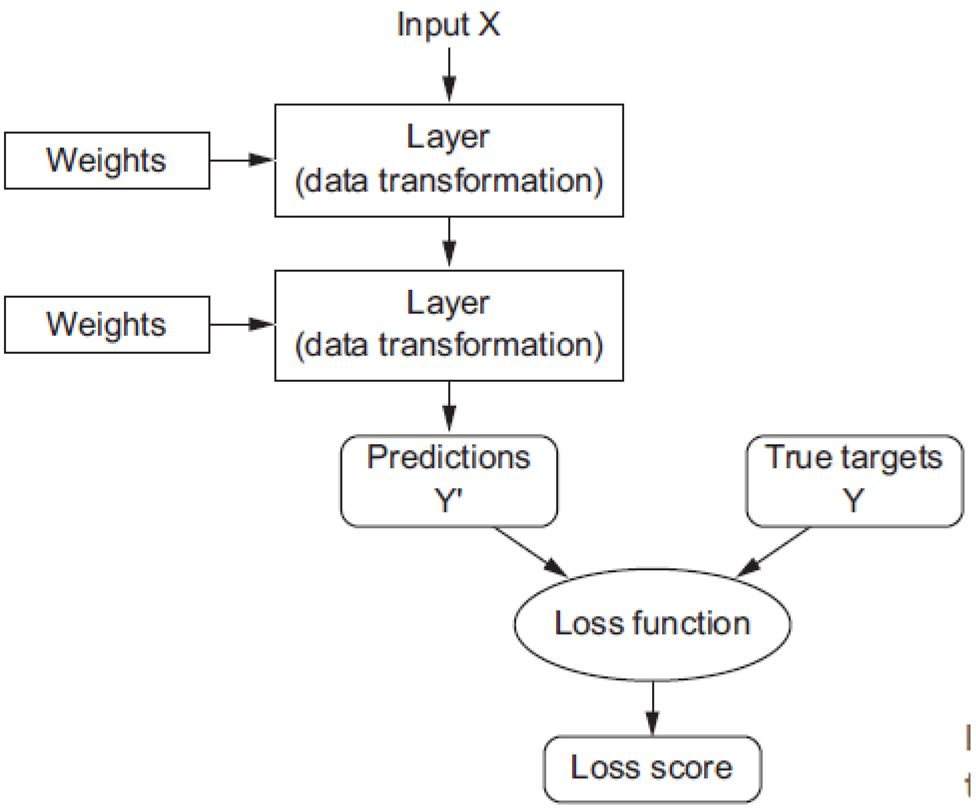
A neural network is parameterized by its weight



Loss Function

- To control something, first you need to be able to observe it. To control the output of a neural network, you need to be able to measure how far this output is from what you expected. This is the job of the *loss function* of the network, also called the *objective function*. The loss function takes the predictions of the network and the true target (what you wanted the network to output) and computes a distance score, capturing how well the network has done on this specific example

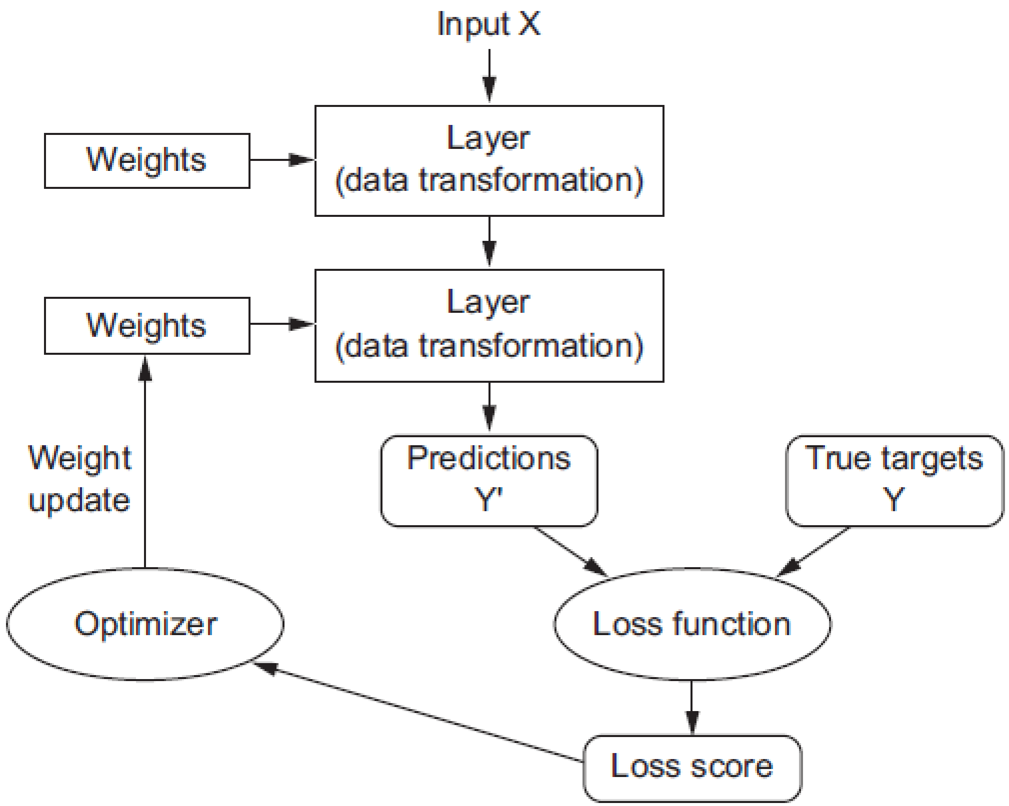
Loss Function



Optimizer/Backpropogation

The fundamental trick in deep learning is to use this score as a feedback signal to adjust the value of the weights a little, in a direction that will lower the loss score for the DIAGRAM. This adjustment is the job of the *optimizer*, which implements what’s called the *Backpropagation* algorithm: the central algorithm in deep learning.

Optimizer/Backprogpogation



The loss score is used as a feedback signal to adjust the weights

Loss Function

The weights of the network are assigned random values, so the network merely implements a series of random transformations. Naturally, its output is far from what it should ideally be, and the loss score is accordingly very high. But with every example the network processes, the weights are adjusted a little in the correct direction, and the loss score decreases.

Loss Function

This is the training loop, which, repeated a sufficient number of times (typically tens of iterations over thousands of examples), yields weight values that minimize the loss function. A network with a minimal loss is one for which the outputs are as close as they can be to the targets: a trained network.

Artificial Neural Networks

Father of AI

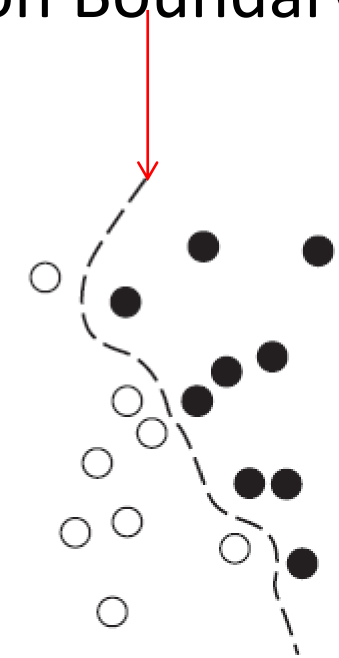
- Geoffrey Hinton at the University of Toronto,
- Yoshua Bengio at the University of Montreal, Yann LeCun at New York University,
- IDSIA in Switzerland.

Kernel Method Failed in Image Classification

Kernel methods are a group of classification algorithms, the best known of which is the *support vector machine* (SVM).

SVMs exhibited state-of-the-art performance on
simple classification problems

However SVMs failed in image classification



Gradient Boosting popular in ML algorithms

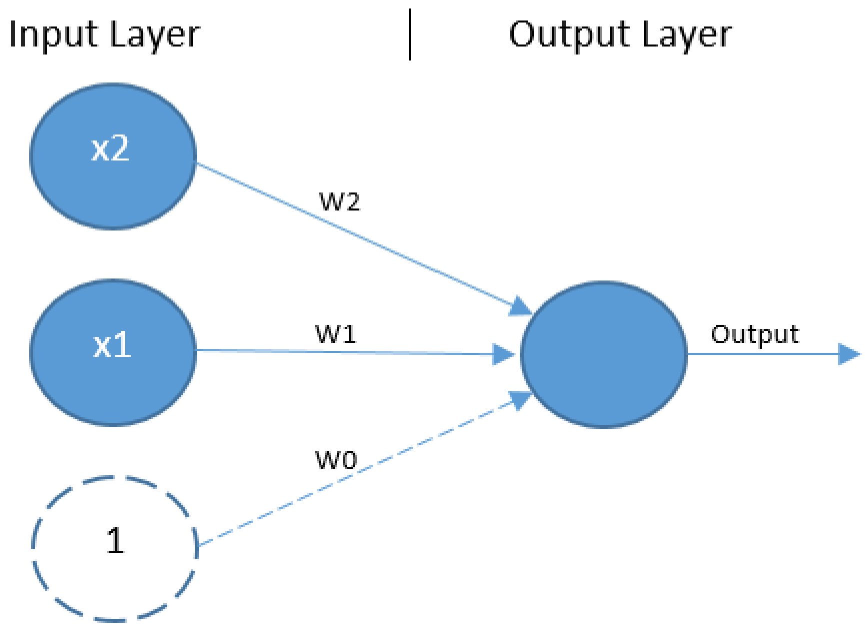
After 2014 *gradient boosting machines* took over. A gradient boosting machine, much like a random forest, is a machine-learning

Technique based on ensembling weak prediction models, generally decision trees.

- Gradient Boosting is the best algorithm for dealing with nonperceptual data today.

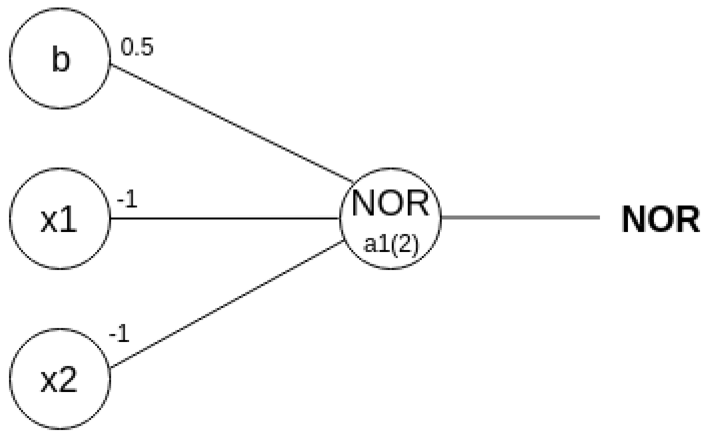
Neural Network algorithms

Perceptrons



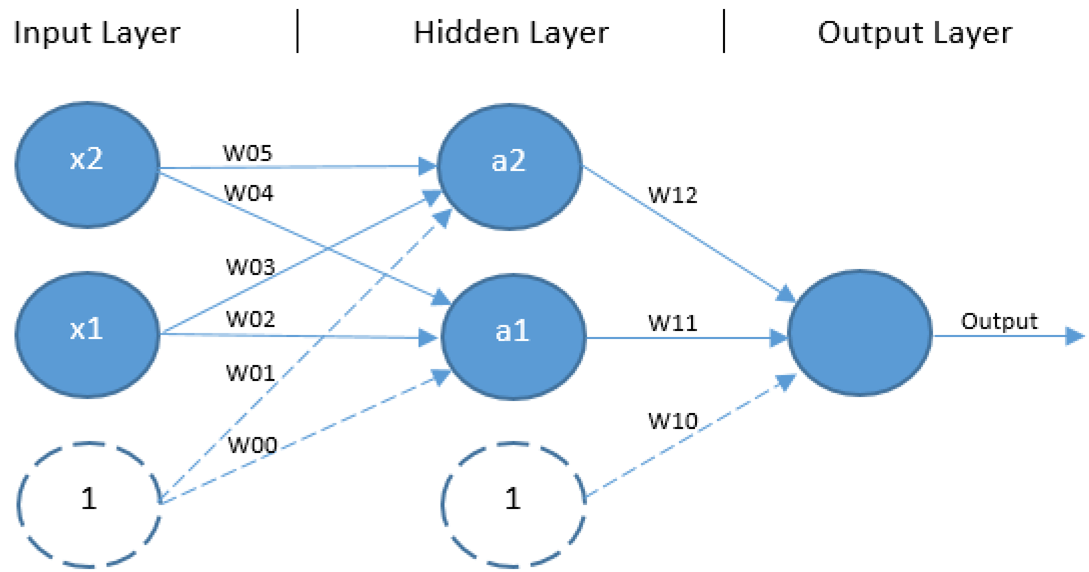
The perceptron is a type of feed-forward network, which means the process of generating an output—known as forward propagation—flows in one direction from the input layer to the output layer. There are no connections between units in the input layer. Instead, all units in the input layer are connected directly to the output unit.

Perceptrons



What are the weights and bias for the NOR perceptron?

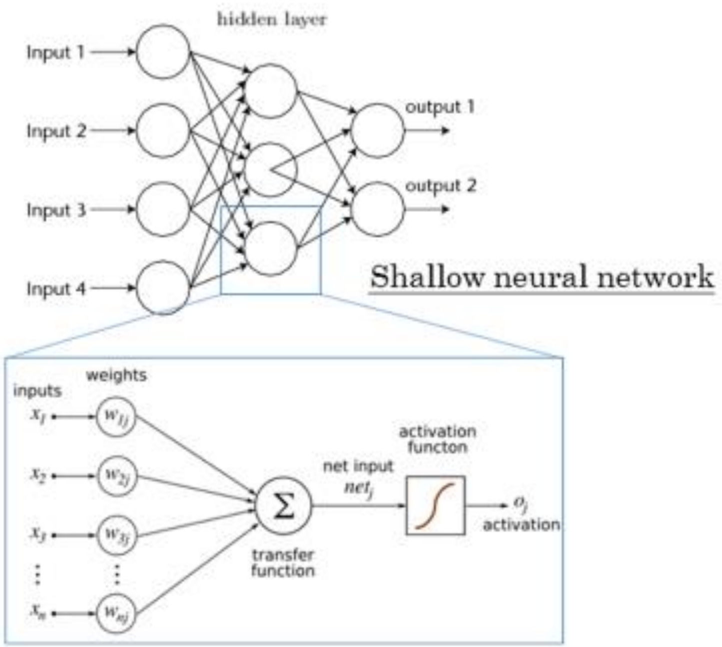
Multilayer Perceptrons

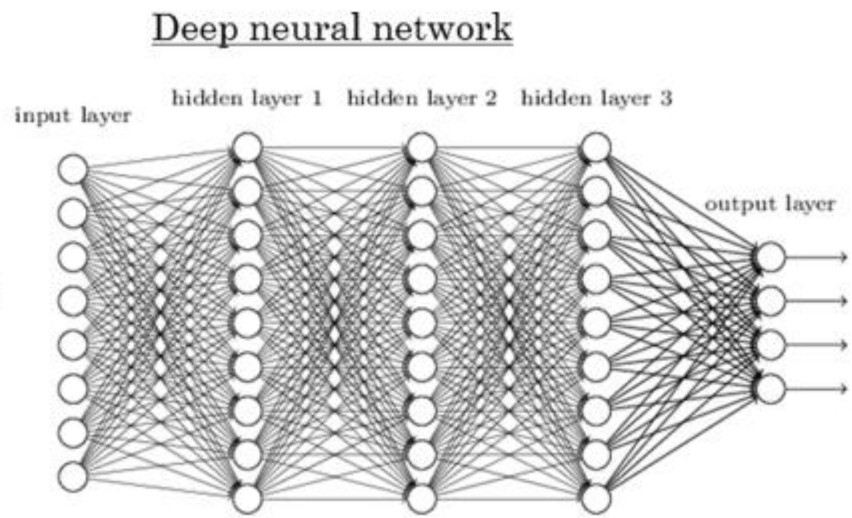


Multilayer Perceptrons

The solution to this problem is to expand beyond the single-layer architecture by adding an additional layer of units without any direct access to the outside world, known as a hidden layer. This kind of architecture—shown in Figure 4—is another feed-forward network known as a multilayer perceptron (MLP).

Deep Neural Network

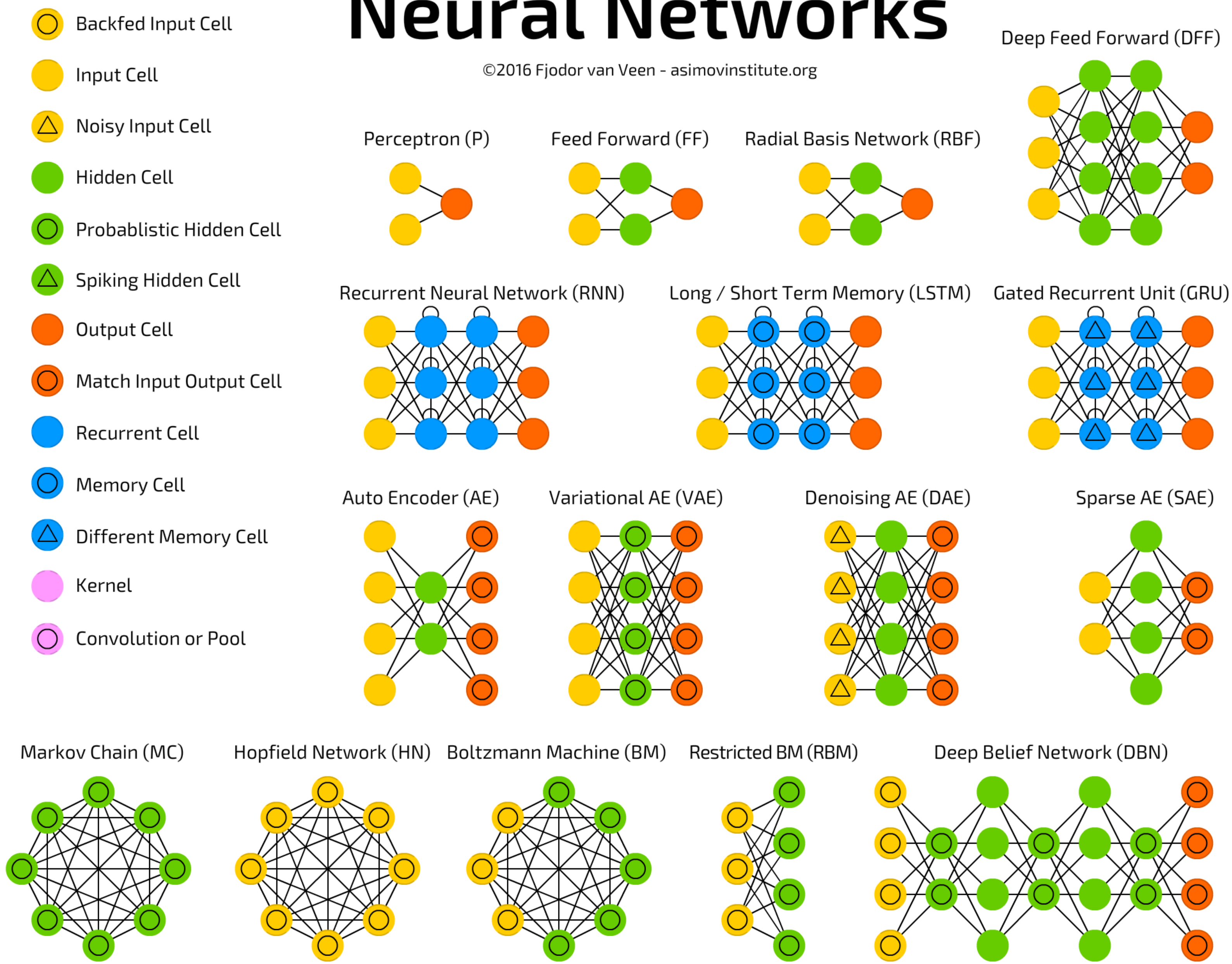




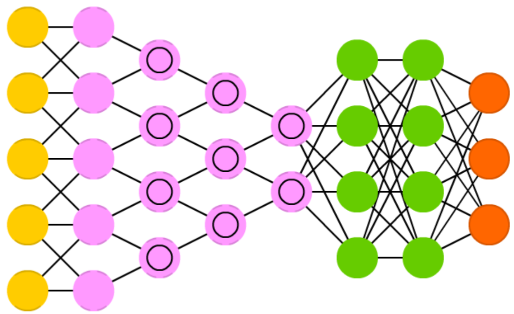
A **deep neural network** (DNN) is an artificial **neural network** (ANN) with multiple layers between the input and output layers.

A mostly complete chart of

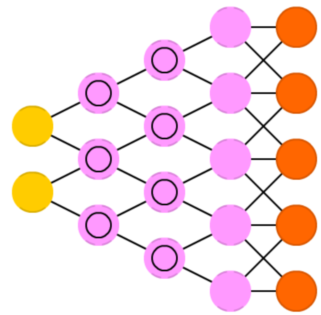
Neural Networks



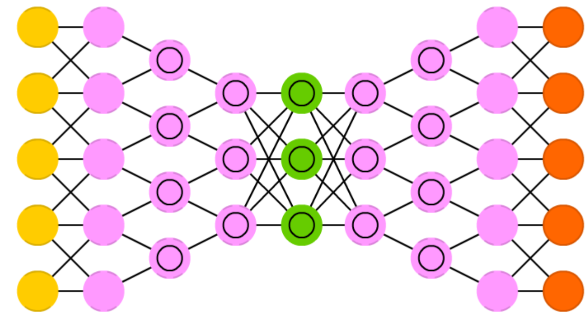
Deep Convolutional Network (DCN)



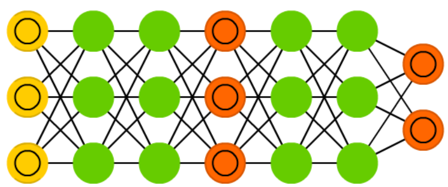
Deconvolutional Network (DN)



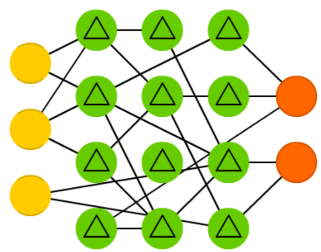
Deep Convolutional Inverse Graphics Network (DCIGN)



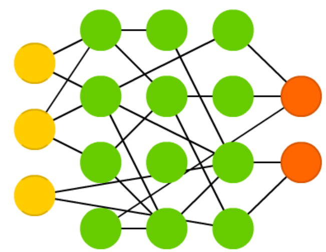
Generative Adversarial Network (GAN)



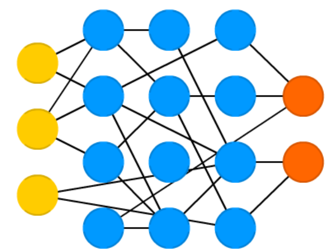
Liquid State Machine (LSM)



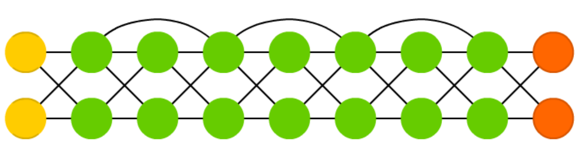
Extreme Learning Machine (ELM)



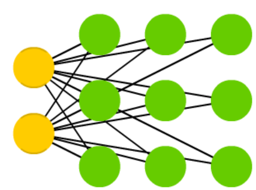
Echo State Network (ESN)



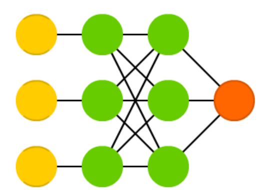
Deep Residual Network (DRN)



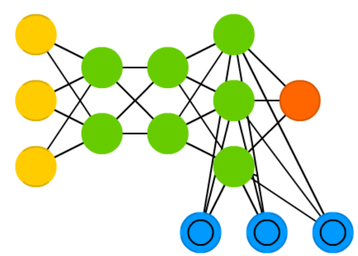
Kohonen Network (KN)



Support Vector Machine (SVM)



Neural Turing Machine (NTM)



Tensors

- Data are stored in multidimensional Numpy
- Arrays, also called *tensors*.
- At its core, a tensor is a container for data—almost always numerical data. So, it’s a container for numbers. You may be already familiar with matrices, which are 2D tensors:
- Tensors are a generalization of matrices to an arbitrary number of dimensions
- In the context of Tensors, a *dimension* is often called an *axis*.

Example of Neural Netowrk

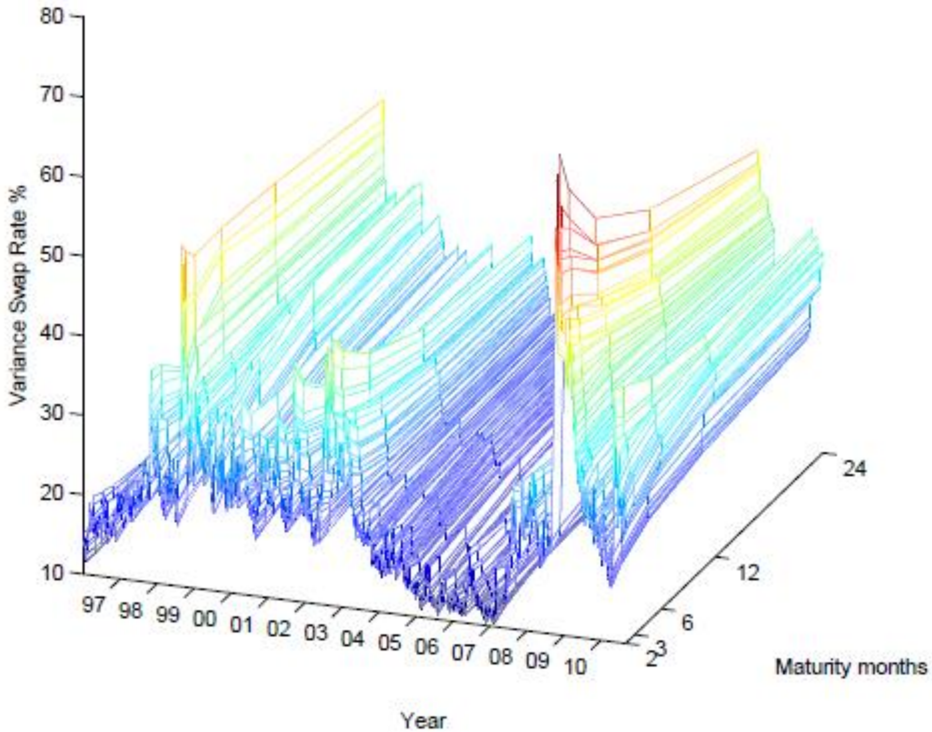
Vectorization of Data

Vectorization

Once we have converted our text samples into sequences of words, we need to turn these sequences into numerical vectors.

3-d vector Matrix

A (224x1) vector with dates (x-axis), a (10x1) vector with maturities (y-axis) and a (224x10) matrix with the values (z-axis).

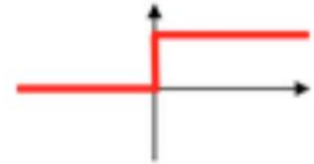
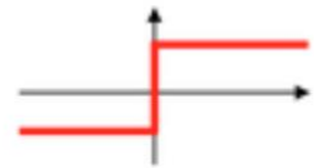
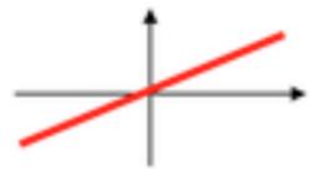
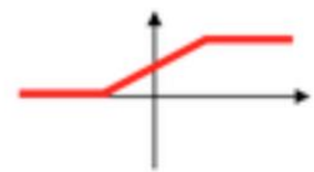
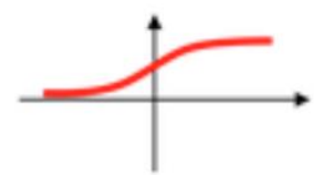
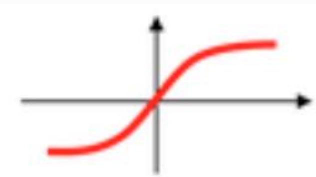


What is an Activation Function

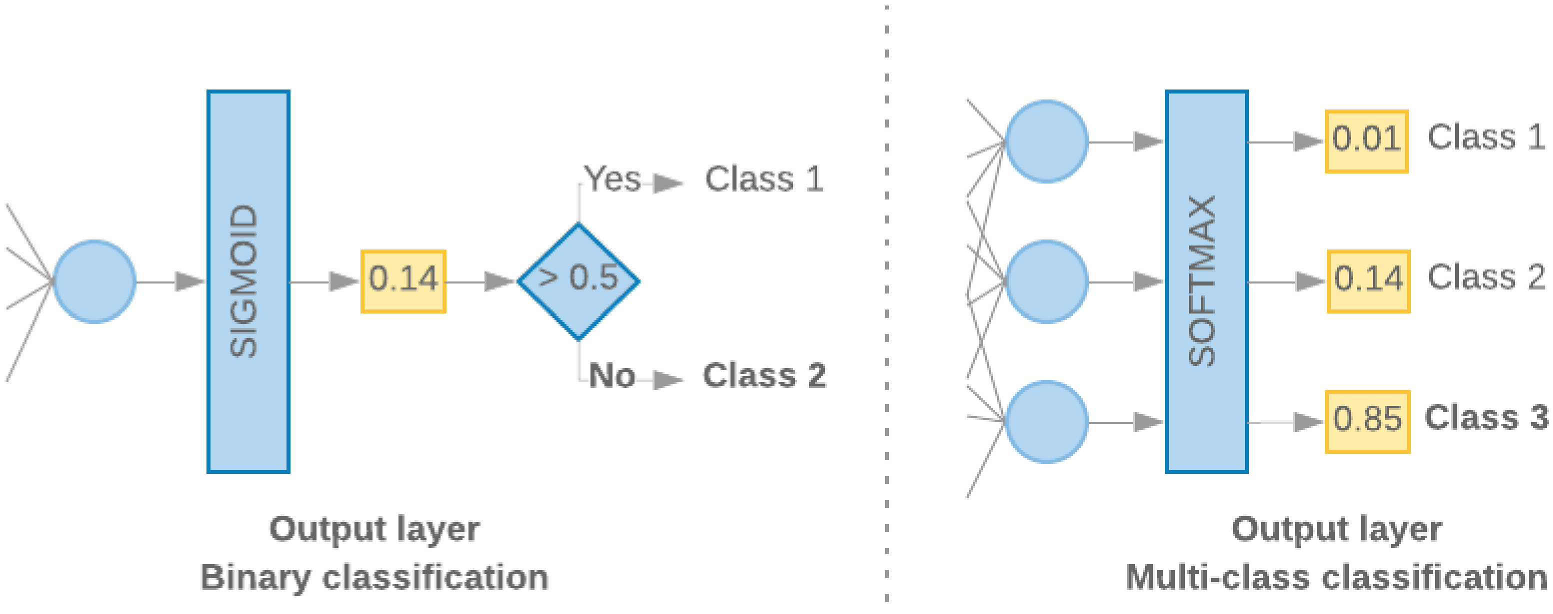
An activation function transforms the shape/representation of the data going into it.

A simple example could be max(0,xi) , a function which outputs 0 if the input xi is negative or xi if the input xi is positive. This function is known as the “ReLU” or “Rectified Linear Unit” activation function.

The choice of which function(s) depends on the problem we are solving.

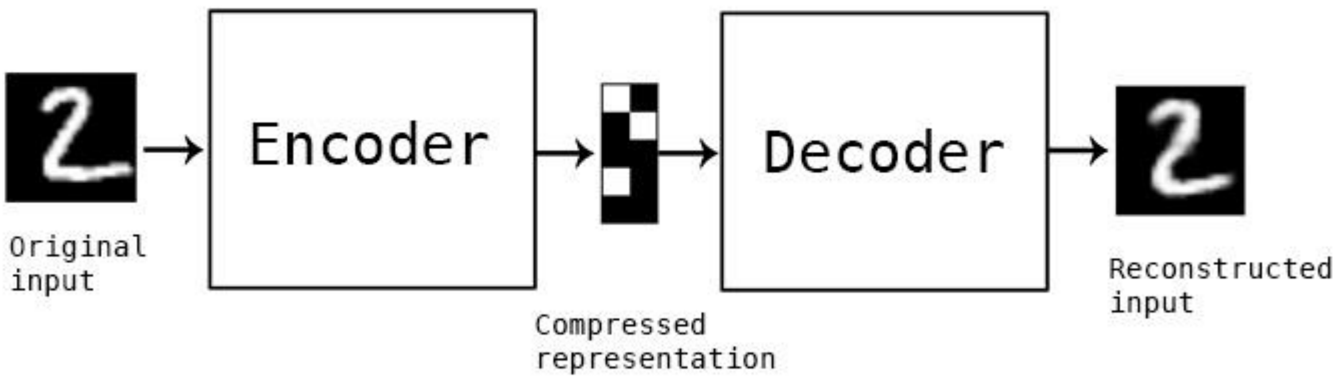
Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multi-layer NN	

Which Activation Function to use?

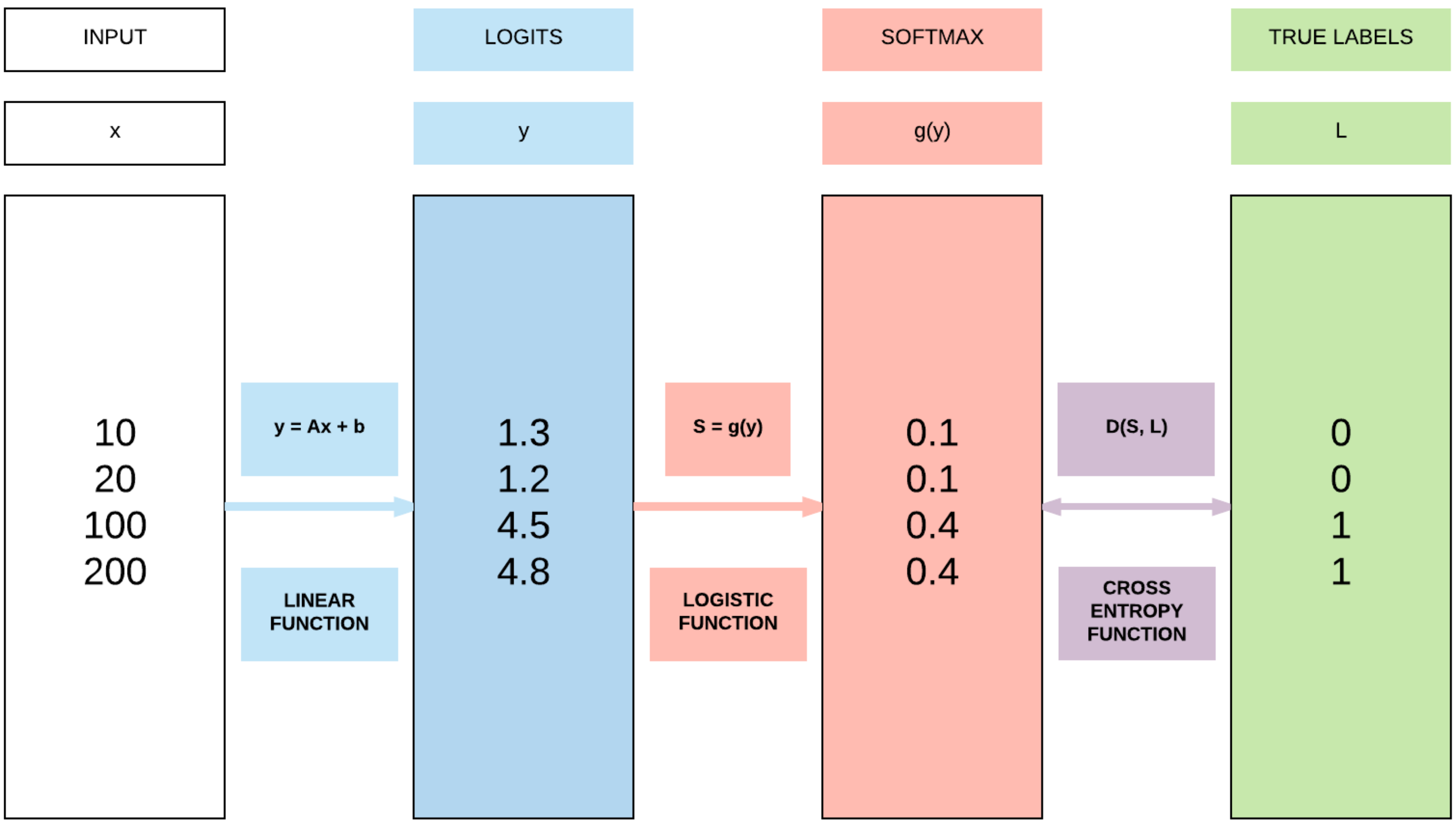


Auto Encoders

- "Autoencoding" is a data compression algorithm where the compression and decompression functions are data-specific,



Deep Learning FLow



MINST

Deep Learning Example

20 Aug 2020

60K Train Images

60K Labels Image 2d

$\boxed{1}$ 1

$\boxed{9}$ 9

$\boxed{3}$ 3

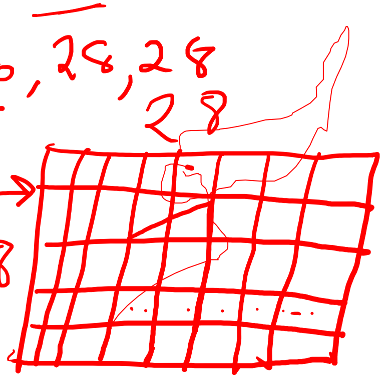
Shape 3d

60000, 28, 28

28

$\boxed{1}$ →

28



MINST – Business Use case

The problem we're trying to solve here is to classify grayscale images of handwritten

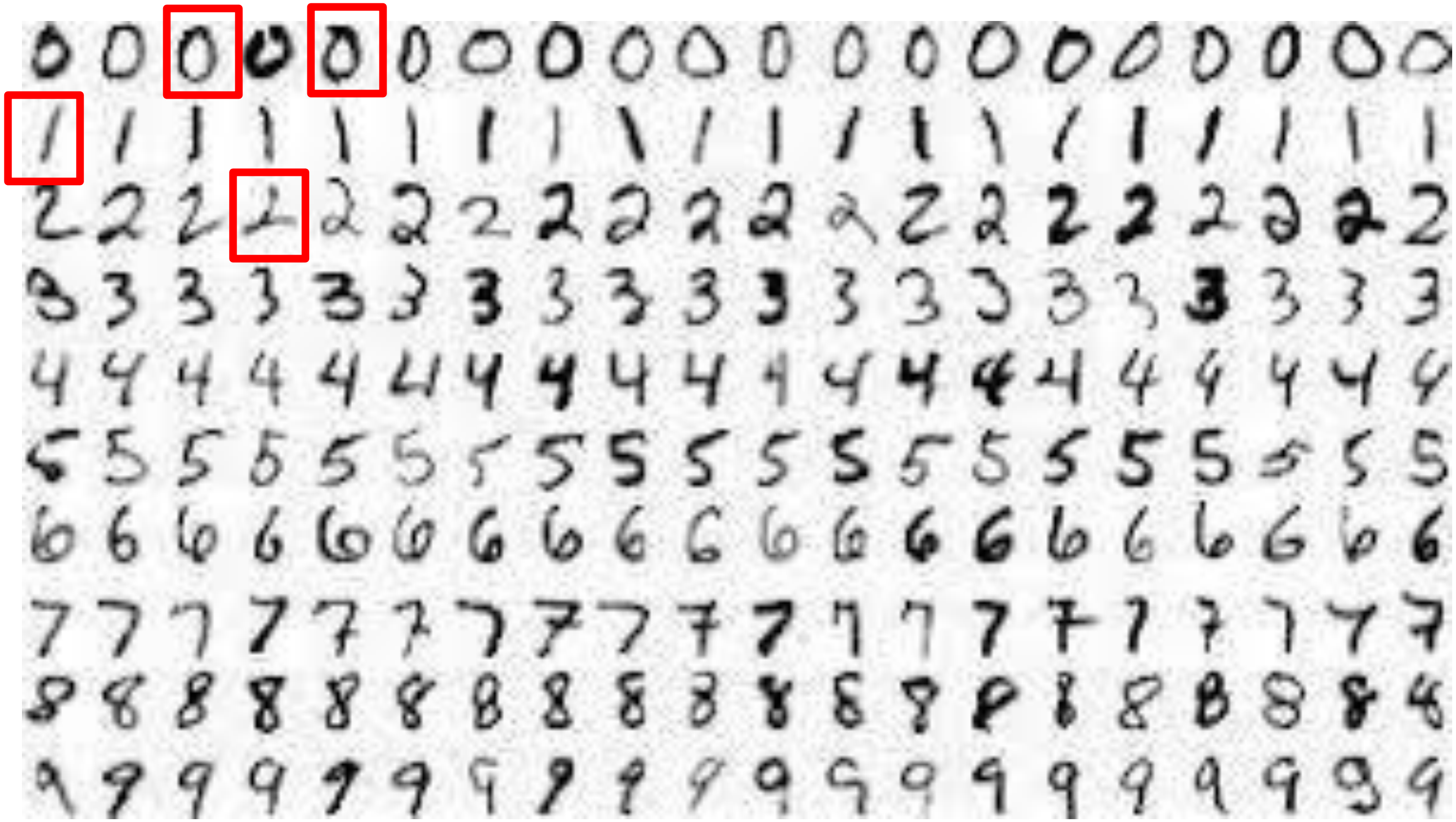
digits (28 × 28 pixels) into their 10 categories (0 through 9). Identify the numbers of Handwritten digits.

We'll use the MNIST. It's a set of 60,000 training images, plus 10,000 test images, assembled by the National Institute of Standards and Technology (the NIST in MNIST) in the 1980s.

How MINST Data Images data is converted to Numeric Data

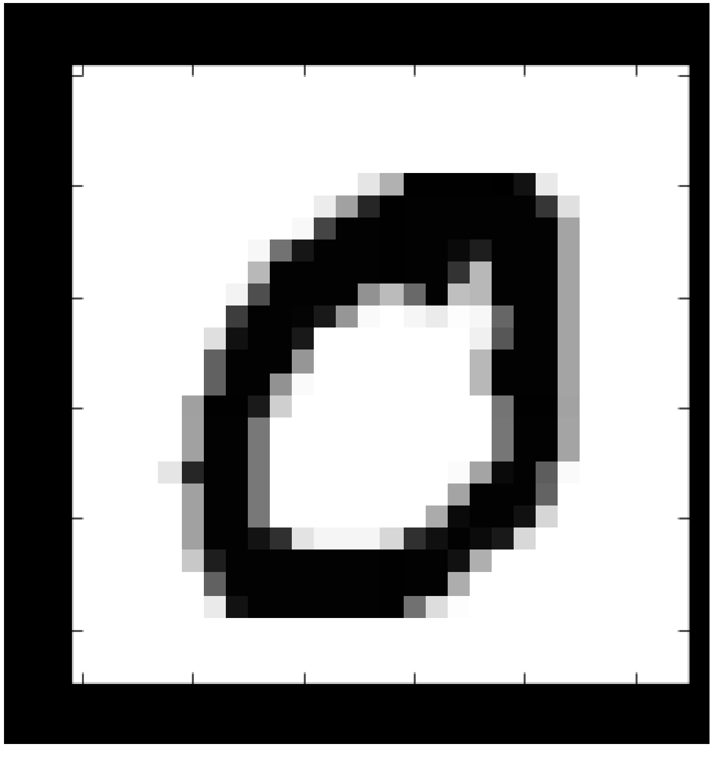
2neural-network-Number Detection.ipynb

28x28 RGB 1 channel 1 So the matrix is data is not having all RGB values instead it is having only 1 Value

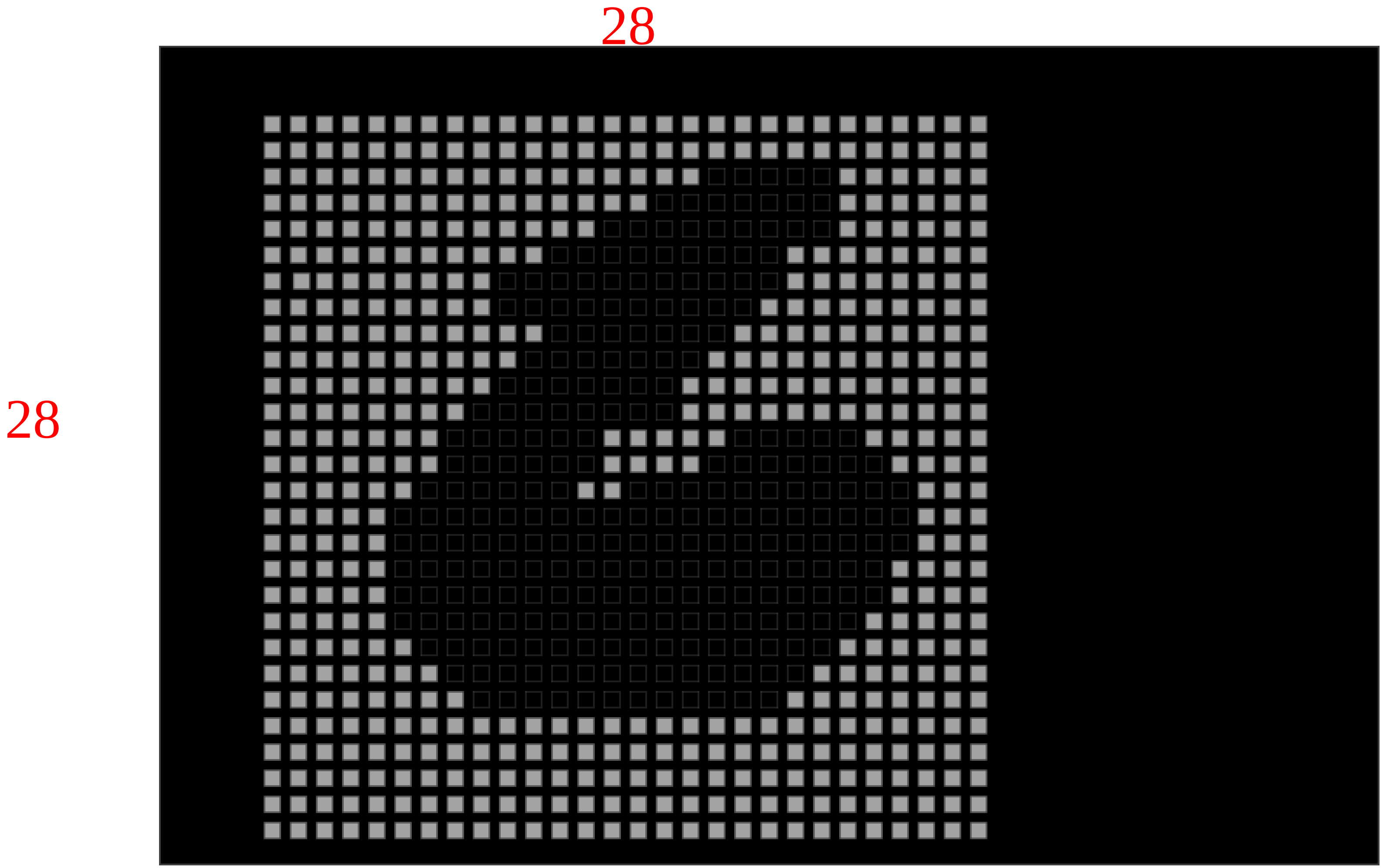


60000

zero

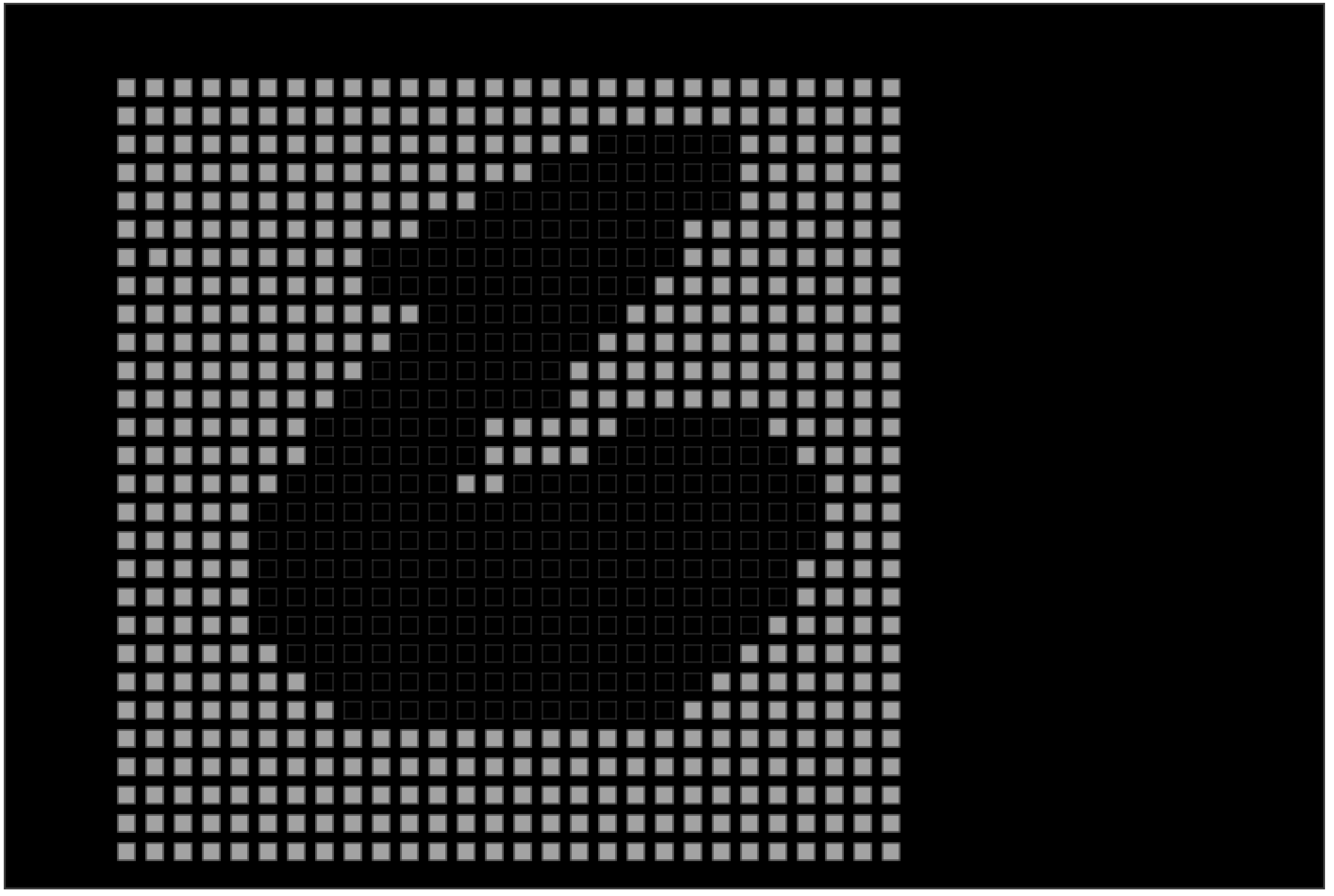


How Image 6 is converted

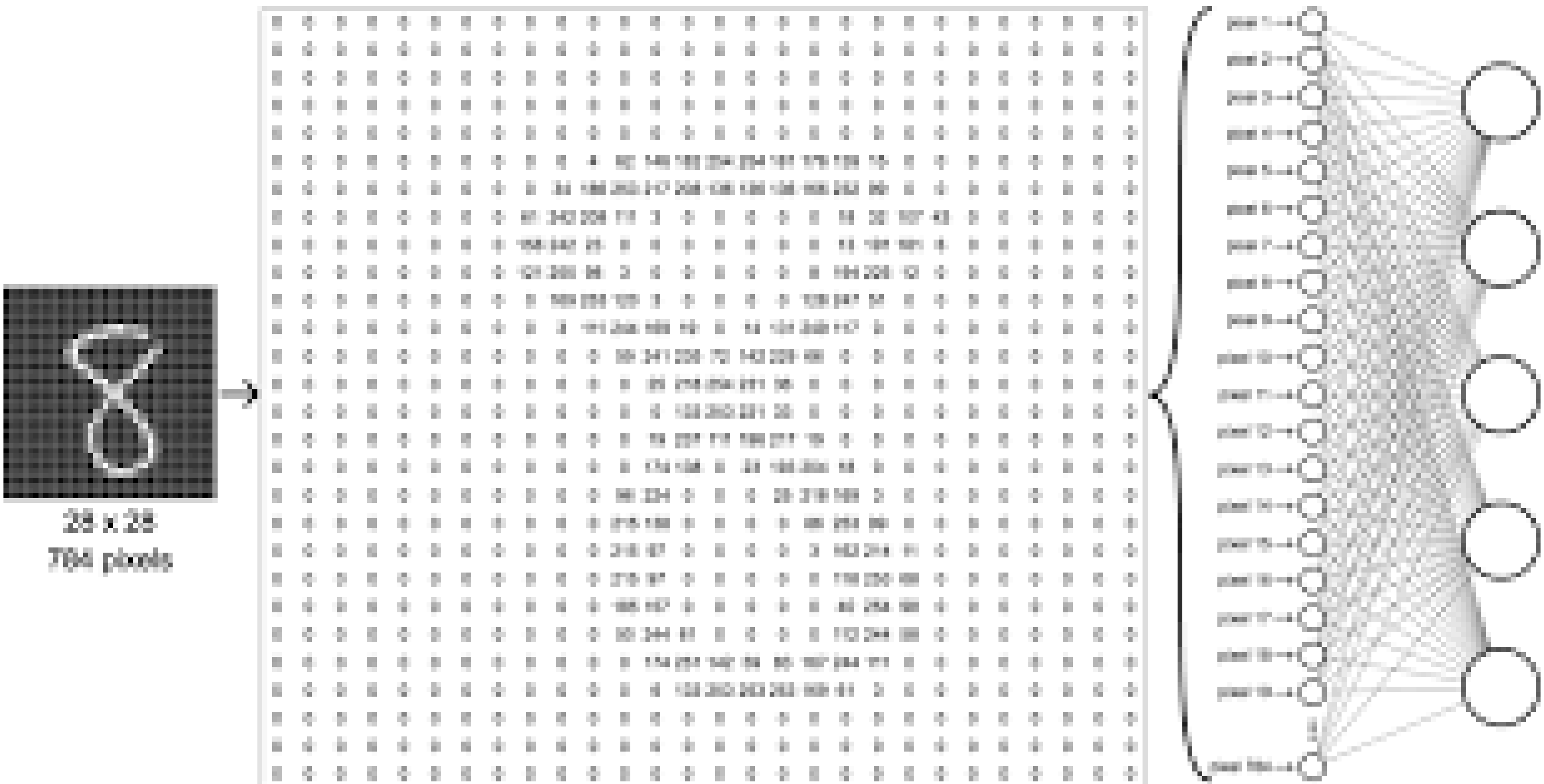


0 to 255 RGB value for Black

How Image 6 is Converted



How Image 8 Is converted



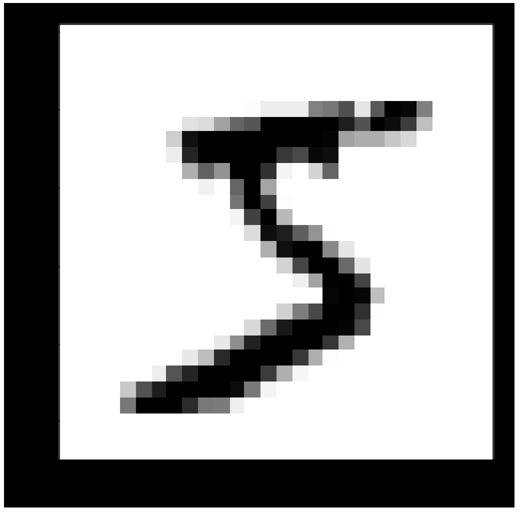
How the Image is feed to a Neural Network – Image 5

train_images[0]

```
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   3,
         18,  18,  18, 126, 136, 175,  26, 166, 255, 247, 127,   0,   0,
          0,   0],
```

Label Image 5

- train_labels[0] =>5
- Convert the Matrix data to 5



Sequential, Functional

- The **sequential** API allows you to create models layer-by-layer for most problems. It is limited in that it does not allow you to create models that share layers or have multiple inputs or outputs.
- The **functional** API allows you to create models that have a lot more flexibility as you can easily define models where layers connect to more than just the previous and next layers

keras.layers.Dense(units, activation=**None**

- keras.layers.Dense(units, activation=**None**, use_bias=**True**, kernel_initializer='glorot_uniform', bias_initializer='zeros', kernel_regularizer=**None**, bias_regularizer=**None**, activity_regularizer=**None**, kernel_constraint=**None**, bias_constraint=**None**)

keras.layers.Dense(units, activation=**None**

- *# as first layer in a sequential model:* model = Sequential() model.add(Dense(32, input_shape=(16,)))
- *# now the model will take as input arrays of shape (*, 16)*
- *# and output arrays of shape (*, 32) # after the first layer, you don't need to specify # the size of the input anymore:* model.add(Dense(32))

https://keras.io/layers/core/

- keras.layers.Dense(units, activation=**None**, use_bias=**True**, kernel_initializer='glorot_uniform', bias_initializer='zeros', kernel_regularizer=**None**, bias_regularizer=**None**, activity_regularizer=**None**, kernel_constraint=**None**, bias_constraint=**None**)

Networks = Models.sequential

- There are two ways to build **Keras models**: **sequential** and functional. The **sequential** API allows you to create **models** layer-by-layer for most problems. It is limited in that it does not allow you to create **models** that share layers or have multiple inputs or outputs

Dense Layers and Dropout-1

- **from** keras.layers **import** Sequential
- model = Sequential()
- model.add(Dropout (0.2))
- model.add(Dense(10, activation = 'relu')

Dense Layers and Dropout-2

- A dense layer is just a regular layer of neurons in a neural network. Each neuron recieves input from all the neurons in the previous layer, thus densely connected. The layer has a weight matrix **W,** a bias vector **b,** and the activations of previous layer **a.** The following is te docstring of class Dense from the keras documentation:

Dense Layers and Dropout-3

- output = activation(dot(input, kernel) + bias)where activation is the element-wise activation function passed as
  the activation argument, kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer.

Dense and Dropout -4

- Dropout is a a technique used to tackle Overfitting . The Dropout method in keras.layers module takes in a float between 0 and 1, which is the fraction of the neurons to drop. Below is the docstring of the Dropout method from the documentation:
- Dropout consists in randomly setting a fraction rate of input units to 0 at each update during training time, which helps prevent overfitting.

Uses Cases for DL

Apply Deep Learning to the three new problems covering the three most common use cases of neural networks:

- binary classification,
- multiclass classification
- scalar regression.

```
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255
```

- Before training, we'll preprocess the data by reshaping it into the shape the network
- expects and scaling it so that all values are in the [0, 1] interval. Previously, our training
- images, for instance, were stored in an array of shape (60000, 28, 28) of type

```
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255
```

- uint8 with values in the [0, 255] interval. We transform it into a float32 array of
- shape (60000, 28 * 28) with values between 0 and 1.

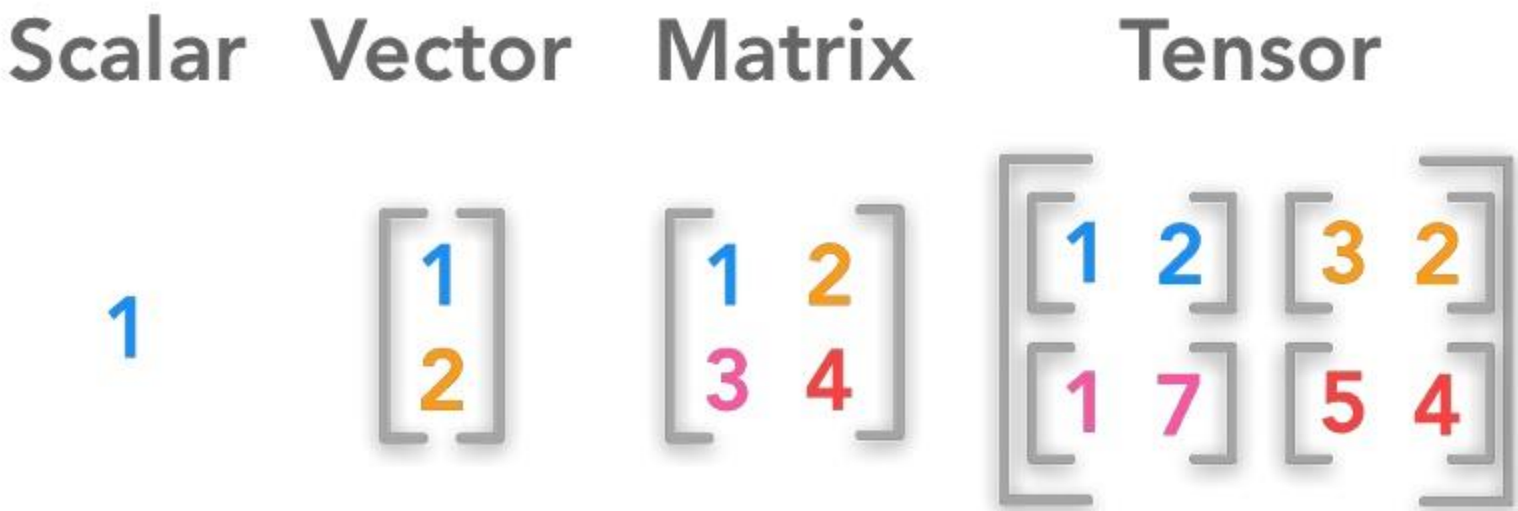
To_categorical(train_labels)

- To vectorize the labels, there are two possibilities: you can cast the label list as an integer
- tensor, or you can use one-hot encoding. One-hot encoding is a widely used format
- for categorical data, also called *categorical encoding*.

To_categorical(train_labels)

- explanation of one-hot encoding, see section 6.1. In this case, one-hot encoding of
- the labels consists of embedding each label as an all-zero vector with a 1 in the place of
- the label index.

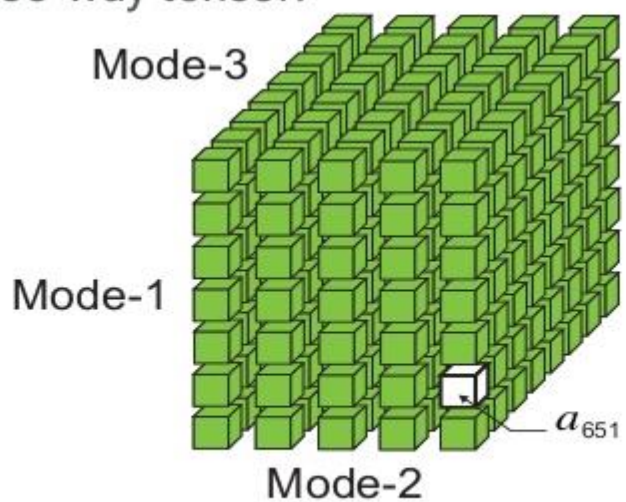
Data types



Tensor

WHAT IS A TENSOR?

A tensor is a multidimensional array
E.g., three-way tensor:



IMDB

Deep Learning Example

Preparing the IMDB data

- ***Tokenization***: Divide the texts into words or smaller sub-texts, which will enable good generalization of relationship between the texts and the labels. This determines the "vocabulary" of the dataset (set of unique tokens present in the data).
- ***Vectorization***: Define a good numerical measure to characterize these texts.

IMDB Business Use Case

Categorize a set of highly polarized reviews from the Internet Movie Database to Positive or Negative Review

IMDB Dataset

- A set of 50,000 highly polarized reviews from the Internet Movie Database.
- They're split into 25,000 reviews for training and 25,000 reviews for testing, each set consisting of 50% negative and 50% positive reviews.
- Why use **separate training and test sets?** Because you should never test a machine learning model on the same data that you used to train it!

IMDB Dataset

- The IMDB dataset comes packaged with Keras. It has already been pre-processed:
- The reviews (sequences of words) have been turned into sequences of integers, where each integer stands for a specific word in a dictionary.

IMBD Data manipulation

- Dataset of 25,000 movies reviews from IMDB, are labelled by sentiment (positive/negative) (1/0)
- Reviews have been preprocessed, and each review is encoded as a sequence of word indexes (integers).
- For convenience, words are indexed by overall frequency in the dataset, so that for instance the integer "3" encodes the 3rd most frequent word in the data.
- This allows for quick filtering operations such as: "only consider the top 10,000 most common words, but eliminate the top 20 most common words".

Structure of IMDB Data

```
print('---review---')
print(X_train[6])
print('---label---')
print(y_train[6])
```

```
---review---
[1, 2, 365, 1234, 5, 1156, 354, 11, 14, 2, 2, 7, 1016, 2, 2, 356, 44, 4, 1349, 500, 746, 5, 200, 4, 4132, 11, 2, 2, 1117, 1831,
2, 5, 4831, 26, 6, 2, 4183, 17, 369, 37, 215, 1345, 143, 2, 5, 1838, 8, 1974, 15, 36, 119, 257, 85, 52, 486, 9, 6, 2, 2, 63, 27
1, 6, 196, 96, 949, 4121, 4, 2, 7, 4, 2212, 2436, 819, 63, 47, 77, 2, 180, 6, 227, 11, 94, 2494, 2, 13, 423, 4, 168, 7, 4, 22,
5, 89, 665, 71, 270, 56, 5, 13, 197, 12, 161, 2, 99, 76, 23, 2, 7, 419, 665, 40, 91, 85, 108, 7, 4, 2084, 5, 4773, 81, 55, 52,
1901]
---label---
1
```

Structure of IMDB Data

- Note that the **Review** is stored as a sequence of integers. These are word IDs that have been pre-assigned to individual words, and the **Label** is an integer (0 for negative, 1 for positive).

Converting Reviews to Characters

We can use the dictionary returned by **imdb.get_word_index()** to map the review back to the original words.

```
word2id = imdb.get_word_index()
id2word = {i: word for word, i in word2id.items()}
print('---review with words---')
print([id2word.get(i, ' ') for i in X_train[6]])
print('---label---')
print(y_train[6])
```

Converting Reviews (Integers to Words)

```
---review---
[1, 2, 365, 1234, 5, 1156, 354, 11, 14, 2, 2, 7, 1016, 2, 2, 356, 44, 4, 1349, 500, 746, 5, 200, 4, 4132, 11, 2, 2, 1117, 1831,
2, 5, 4831, 26, 6, 2, 4183, 17, 369, 37, 215, 1345, 143, 2, 5, 1838, 8, 1974, 15, 36, 119, 257, 85, 52, 486, 9, 6, 2, 2, 63, 27
1, 6, 196, 96, 949, 4121, 4, 2, 7, 4, 2212, 2436, 819, 63, 47, 77, 2, 180, 6, 227, 11, 94, 2494, 2, 13, 423, 4, 168, 7, 4, 22,
5, 89, 665, 71, 270, 56, 5, 13, 197, 12, 161, 2, 99, 76, 23, 2, 7, 419, 665, 40, 91, 85, 108, 7, 4, 2084, 5, 4773, 81, 55, 52,
1901]
---label---
1
```

```
---review with words---
['the', 'and', 'full', 'involving', 'to', 'impressive', 'boring', 'this', 'as', 'and', 'and', 'br', 'villain', 'and', 'and', 'n
eed', 'has', 'of', 'costumes', 'b', 'message', 'to', 'may', 'of', 'props', 'this', 'and', 'and', 'concept', 'issue', 'and', 't
o', "god's", 'he', 'is', 'and', 'unfolds', 'movie', 'women', 'like', "isn't", 'surely', "i'm", 'and', 'to', 'toward', 'in', "he
re's", 'for', 'from', 'did', 'having', 'because', 'very', 'quality', 'it', 'is', 'and', 'and', 'really', 'book', 'is', 'both',
'too', 'worked', 'carl', 'of', 'and', 'br', 'of', 'reviewer', 'closer', 'figure', 'really', 'there', 'will', 'and', 'things',
'is', 'far', 'this', 'make', 'mistakes', 'and', 'was', "couldn't", 'of', 'few', 'br', 'of', 'you', 'to', "don't", 'female', 'th
an', 'place', 'she', 'to', 'was', 'between', 'that', 'nothing', 'and', 'movies', 'get', 'are', 'and', 'br', 'yes', 'female', 'j
ust', 'its', 'because', 'many', 'br', 'of', 'overly', 'to', 'descent', 'people', 'time', 'very', 'bland']
---label---
1
```

Data Manipulation for RNN

Padding

In order to feed this data into our RNN, all input documents must have the same length. We will limit the maximum review length to max_words by truncating longer reviews and **padding** shorter reviews with a null value (0). We can accomplish this using the **pad_sequences()** function in Keras. For now, set max_words to 500.

Assigning an index in Dictionary

Texts: 'The mouse ran up the clock'

Dictionary:
[The :1 , mouse : 2, ran: 3, up: 4, clock: 5....}

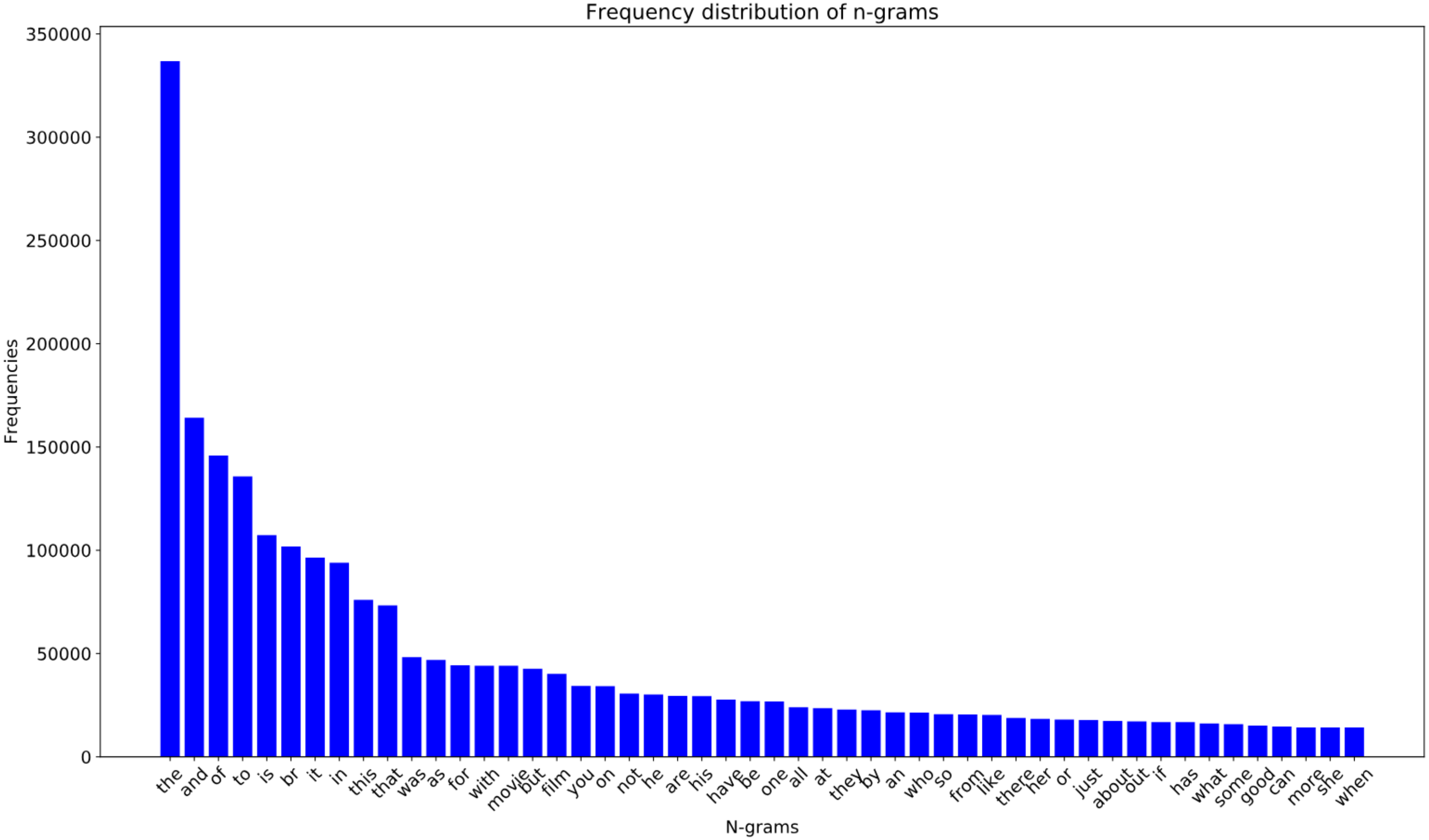
Index assigned for every token:

{'clock': 5, 'ran': 3, 'up': 4, 'down': 6, 'the': 1, 'mouse': 2}.

'the' occurs most frequently, so the index value of 1 is assigned to it.
Some libraries reserve index 0 for unknown tokens, as is the case here.
Sequence of token indexes:
'The mouse ran up the clock' = [1, 2, 3, 4, 1, 5]

Frequency of Words

Frequency decides the index



One-Hot Encoding

- ***One-hot encoding***: Sequences are represented using word vectors in n- dimensional space where n = size of vocabulary. This representation works great when we are tokenizing as characters, and the vocabulary is therefore small. When we are tokenizing as words, the vocabulary will usually have tens of thousands of tokens, making the one-hot vectors very sparse and inefficient.

One-Hot Encoding

- 'The mouse ran up the clock' =

[[0, 1, 0, 0, 0, 0, 0],

[0, 0, 1, 0, 0, 0, 0],

[0, 0, 0, 1, 0, 0, 0],

[0, 0, 0, 0, 1, 0, 0],

[0, 1, 0, 0, 0, 0, 0],

[0, 0, 0, 0, 0, 1, 0]]

One-Hot Encoding

- 'The mouse ran up the clock' =

The [[0, 1, 0, 0, 0, 0, 0],

mouse[0, 0, 1, 0, 0, 0, 0],

ran[0, 0, 0, 1, 0, 0, 0],

up [0, 0, 0, 0, 1, 0, 0],

the [0, 1, 0, 0, 0, 0, 0],

clock[0, 0, 0, 0, 0, 1, 0]]

Thanks

https://www.linkedin.com/in/arunkumarnair/

Mobile: +91-9890652675

Email: arunkg99@gmail.com