Advanced Data & Network Mining

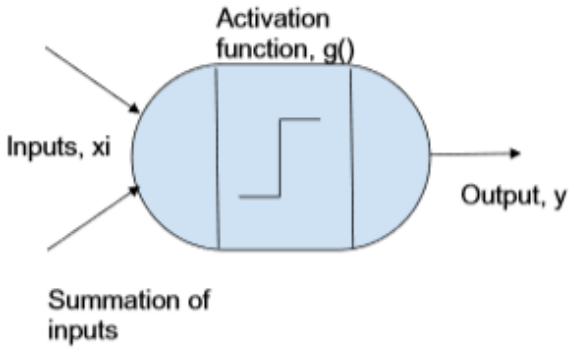
Deep Learning

2023-24

terri.hoare@dbs.ie

Deep Learning
Perceptron (neuron)

The first ANN were the perceptron's developed in the 1950s which were a class of pattern recognition elements that weighed evidence and tested if it exceeded a certain threshold in order to make a decision, that is, to classify patterns. Below the architecture of a single perceptron (later on called neuron) which has retained its basic structure through the years.



Deep Learning
Perceptron (neuron)

Each input, x_i has a weight w_i associated with it and a dot product $\sum w_i x_i$ is computed at the perceptron and passed to the activation function g. If $g(w_i x_i)$ evaluates above a threshold then the output is set to 1 (true) or otherwise to 0 (false).

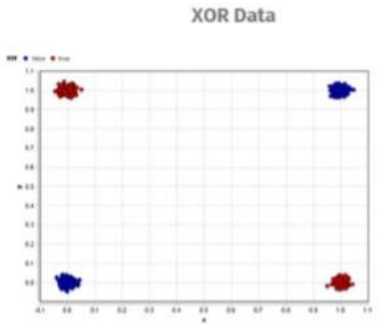
The process of obtaining the weights, w_i is called “learning” or “training” the perceptron. The perceptron learning rule was originally developed by Frank Rosenblatt (1957). Training data are presented to the network’s inputs and the output is computed. The weights w_i are modified by an amount that is proportional to the product of the difference between the actual output, y, and the desired output,d.

Deep Learning
AI Winter – 1970's

Perceptron's were able to solve a range of decision problems, in particular they were able to represent logic gates such as "AND", "OR," and "NOT." The perceptron learning rule tended to converge to an optimal set of weights for several classes of input patterns. However, this was not always guaranteed. Another limitation arose when the data were not linearly separable— for example, the classic "XOR." A XOR gate resolves to "true" if the two inputs are different and resolves to "false" if both inputs are the same (below – perceptron cannot find a hyper-plane that partitions the classes perfectly). Minsky and Papert published these and other core limitations of perceptron's in a 1969 book called Perceptron's, which arguably reduced further interest in these types of ANN and the AI Winter set in.

$$y = x_1 \oplus x_2$$

Row No.	xor	x	y
1	true	1	0
2	false	0	0
3	false	1	1
4	false	0	0
5	true	1	0
6	true	0	1
7	true	1	0
8	false	1	1
9	false	1	1
10	true	1	0
11	false	0	0
12	true	1	0



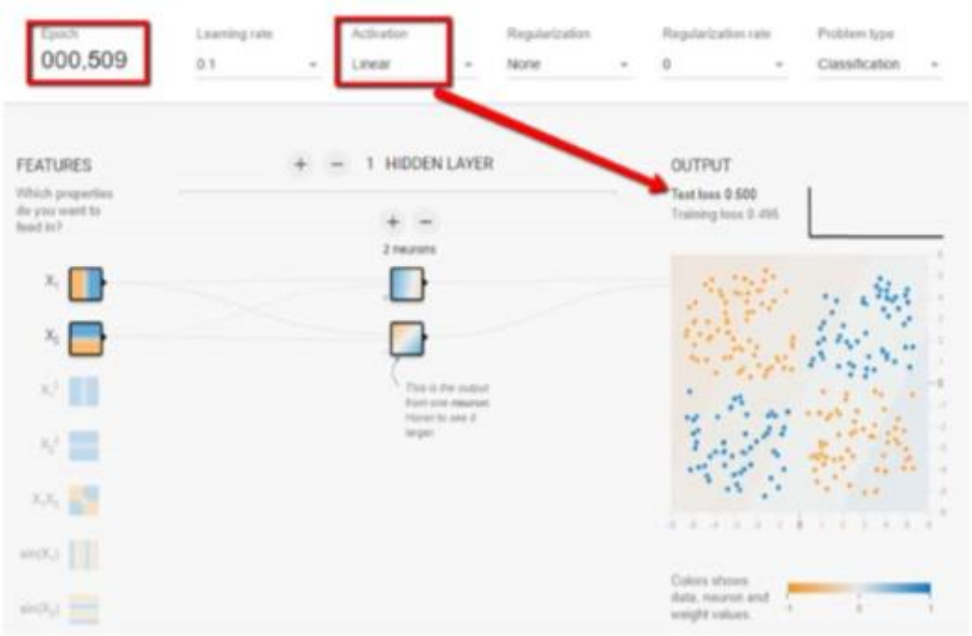
"Data Science Concepts and Practice" 2018 MK : Vijay Kotu and Bala Deshpande

Deep Learning
Mid Winter Thaw of 1980's

ANN, however, had a brief resurgence in the 1980s with the development of the multi-layer perceptron (MLP) which was heralded as the solution for nonlinearly separable functions: for example, changing the activation function in an MLP from a linear step function to a nonlinear type (such as sigmoid) could overcome the decision boundary problem seen in the XOR case.

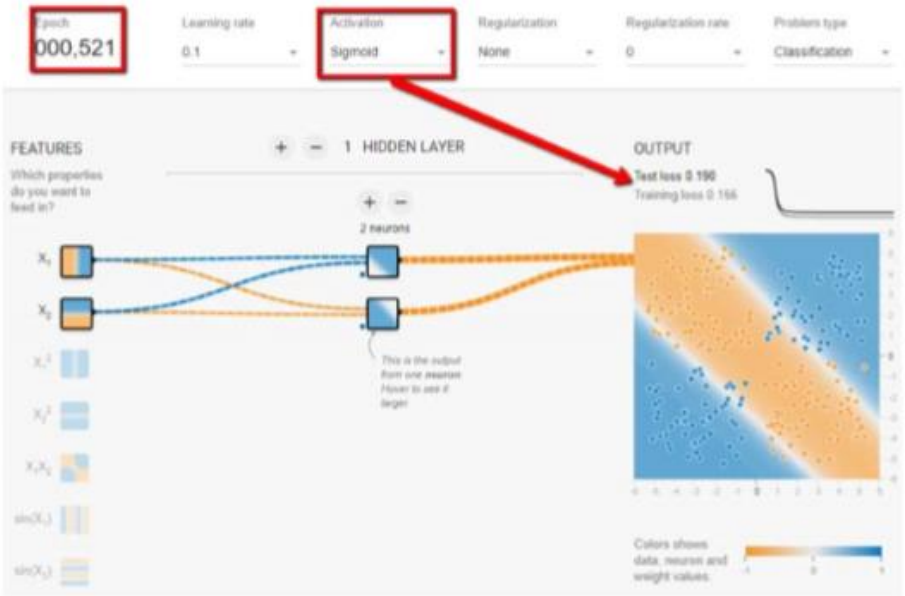
Deep Learning
Mid Winter Thaw of 1980's

Below shows that with a linear activation function a two-layer MLP still fails to achieve more than 50% accuracy on the XOR problem using TensorFlow 4 playground.



Deep Learning
Mid Winter Thaw of 1980's

A simple switch of the activation function to the nonlinear "sigmoid" helps achieve more than 80% accuracy with the same architecture.



Deep Learning
Mid Winter Thaw of 1980's

Another important innovation in the 1980s that was able to overcome some of the limitations of the perceptron training rule was the use of "backpropagation" to calculate or update the weights (rather than reverting back to the inputs every time there was an error.

The perceptron learning rule updated the weights by an amount that was proportional to the error times of the inputs. These weights w_i are the heart of the network and training an multi-layer perceptron (MLP) or ANN is really all about finding these weights. Finding a robust and repeatable process for updating the weights becomes critical. To do this an extra layer of neurons were added in between the input and the output nodes. Now the error quantity becomes a summation and to avoid sign bias, the error may be squared. The challenge now was determining which direction to change the weights, w_i so that this error quantity is minimized.

Deep Learning
Mid Winter Thaw of 1980's

This backpropagation method was introduced by Rumelhart, Hinton, and Williams (1986). Their network was trainable to detect mirror symmetry; to predict one word in a triplet when two of the words were given and other such basic applications.

More sophisticated ANNs were built using backpropagation, that could be trained to read handwriting (LeCun, 1989). However, successful business applications of ANN were still limited and it failed to capture the public imagination the way it has currently.

Part of the reason was the state of computing hardware at the time when these algorithms were introduced. But one can argue that a bigger hurdle preventing a wider adoption back in the 1980s and 1990s was a lack of data. Many of the machine learning algorithms were developed and successfully demonstrated during this time: Hidden Markov Models and Convolutional Neural Nets were described in 1984 and 1989 respectively.

Deep Learning
Mid Winter Thaw of 1980's

However, a successful deployment of these algorithms on a practical business scale did not occur until nearly a decade later. Data (or lack thereof) was the primary reason for this. Data became more readily available and accessible only after the introduction of the internet in 1993.

Wissner-Gross (2016) cites several interesting examples of breakthroughs in AI algorithms, effectively concluding that the average time period for an AI innovation to become practical was 18 years (after the introduction of the algorithm) but only 3 years after the first large scale datasets (that could be used to train that algorithm) became available.

Deep Learning
Mid Winter Thaw of 1980's

In 2006, Hinton (the same Hinton who was part of the team that introduced backpropagation) and Salakhutdinov, demonstrated that by adding more layers of computation to make neural networks "deep," larger datasets could be better utilized to solve problems such as handwriting recognition (Hinton and Salakhutdinov, 2006). These 3-hidden layer deep learning networks had significantly lower errors than the traditional single hidden layer ANNs.

With this innovation, the field of AI emerged from its long winter. This emergence is best summarized in the authors' own words:

"It has been obvious since the 1980s that backpropagation through deep autoencoders would be very effective...provided that computers were fast enough, data sets were big enough, and the initial weights were close enough to a good solution. All three conditions are now satisfied".

Deep Learning
Mid Winter Thaw of 1980's

Using massive datasets, deep network architectures with new and powerful graphics processing units (GPUs) originally developed for video games, real-world AI applications such as facial recognition, speech processing and generation, machines defeating humans at their own board games have become possible.

ANNs moved decisively from the research lab to mainstream media hype.

Deep Learning
Spring and Summer of AI – 2006 - To date

In spite of all these exciting developments, today's AI is still far from being what is considered artificial general intelligence (AGI).

"To get a deep-learning system to recognize a hot dog, you might have to feed it 40 million pictures of hot dogs. To get [a two year old] to recognize a hot dog, you show her a hot dog"

– Data Science Concepts and Practice 2018.

Deep Learning
Spring and Summer of AI – 2006 - To date

Defence Advanced Research Projects Agency (DARPA) has developed a nice classification of the evolution of AI into three “waves” based on the main dimensions which reflect the capabilities of the systems: ability to learn, ability to abstract, and ability to reason.

The **first wave of AI** evolution includes “handcrafted knowledge” systems. These are the expert systems and chess playing programs of the 1980s and 1990s. Humans encode into the machines an ability to make decisions based on input data from a specific domain. In other words, these systems have a limited ability to reason but no ability to learn let alone abstract.

Deep Learning
Spring and Summer of AI – 2006 - To date

Second wave systems include today's machine learning and deep learning systems and are generally systems capable of "statistical learning." The uniqueness of these systems is the ability to separate data into different sets or patterns based on learning by relying on large volumes of data. While a rule engine can be added to these statistical learners, these systems still lack the ability to abstract knowledge.

To clarify this: consider that while a facial recognition system is successful at identifying faces, it cannot explicitly explain why a particular face was categorized as such. On the other hand, a human can explain that a particular face was classified as a man because of the facial hair and body dimensions, for example.

Deep Learning
Spring and Summer of AI – 2006 – To date

In the yet-to-be developed **third wave systems**, AI can not only apply encoded rules and learn from data, but can also explain why a particular data point was classified in a particular way.

This is termed a “contextually adaptive” system. DARPA also calls these systems “Explainable AI” or XAI that “produce more explainable models, while maintaining a high level of learning performance (prediction accuracy).”

A first step toward developing XAI is the integration of the now conventional deep machine learning with reinforcement learning (RL).

- Majority of AI today is machine learning and deep learning.
- Majority of that learning is supervised.
- Majority of that supervised learning is classification.

Deep Learning
How it Works

Traditional machine learning approaches hand engineer a set of features. Deep Learning uses multiple layers to progressively extract higher level features from the raw input. For example, in image processing, lower layers may identify edges, while higher layers may identify the concepts relevant to a human such as digits or letters or faces.

We will explore the connection between conventional machine learning and deep learning.

Firstly, linear regression and how it can be represented using an ANN will be examined more closely and then logistic regression will be discussed to reinforce the similarities between conventional machine learning techniques and deep learning.

ANNs (and deep learning) should be regarded as a mathematical process. An ANN basically creates a mapping of data between outputs and inputs that is established using calculus-based optimization techniques.

Deep Learning
How it Works

In a simplified mathematical format, an ANN can be essentially represented as:

$Y = f(X)$
$Y = f(g(b))\ \ where\ X = g(b)$

where b, X and Y are vectors or more generally, tensors.

In this context, vectors are one dimensional arrays of numbers, matrices are two dimensional arrays and tensors are more general n-dimensional arrays.

Deep Learning
How it Works

The process of training ANN is mostly about finding values for the coefficients, b to complete the mapping. The coefficients are calculated by performing a constrained optimization of an error function (error is the difference between a predicted output $\hat{y}$ and the known output, y). A technique to iteratively perform this optimization is called backpropagation.

A basic difference between deep learning and “shallow” machine learning is in the count of the coefficients, b.

Deep learning deals with weights or coefficient counts in the hundreds of thousands to millions whereas conventional machine learning may deal with a few hundred at best. This enhancement of the numerosity in the computation gives deep learning their significant power to detect patterns within data.

Deep Learning
Regression Models As Neural Networks

The equation,

$Y = f(X)$

$Y = f(g(b))\ \ where\ X = g(b)$

is a vectorized form of,

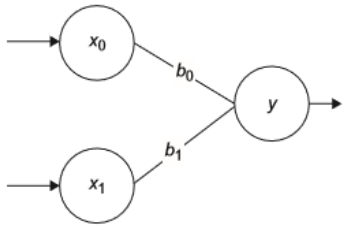
$$y = b_0 + b_1 x_1 + b_2 x_2 + \cdots + b_n x_n$$

the general statement of a multiple linear regression problem.

Databricks Spark https://pages.databricks.com/201809-US-WB-Intro-NeuralNetworks-OnDemand.html discussed how a linear regression problem would be solved using methods of calculus, in particular, gradient descent. The gradient descent technique is the cornerstone of all deep learning algorithms.

Deep Learning
Regression Models as Neural Networks

The linear regression model can be rewritten as an ANN.



Above is the simple linear regression model shown as a network. Note that when n $= 1$, this represents,

$$y = b_0 + b_1 x_1$$

This network is only two layers deep: it has an input layer and an output layer. Multiple regression models simply require additional nodes (one node for each variable/feature) in the input layer and no additional/intermediate layers.

Deep Learning
Regression Models as Neural Networks

Similarly, a logistic regression model can also be represented by a simple two-layer network model with one key difference. As was discussed in Logistic Regression, the output of logistic regression is the probability of an event p rather than a real number value as in linear regression. So, what needed to be done was transform the output variable in such a way that its domain now ranges from 0 to 1 instead of $-\infty$ to $+\infty$. It was observed that by replacing the right-hand side the expression with the log of the odds-ratio or the logit, this transformation was achieved.

$$\log\left(\frac{p}{1-p}\right) = b_0 + b_1 x_1$$

$$and\ p = \left(\frac{1}{1+e^{-z}}\right)\ \ where\ z = b_0 + b_1 x_1$$

or $more\ generally\ p(y) = \sigma(z)\ (sigmoid\ function)$

Deep Learning
Regression Models as Neural Networks

The sigmoid's domain is $[0,1]\ for\ z \in [-\infty, +\infty]$ so that any arbitrary values for $b\ and\ x$ will always result in $p(y_n) = [0,1]$. Note that $p(y_n)$ is the prediction from the logistic regression model for sample n, which needs to be compared with the actual class value, p_n for that sample, in order to evaluate the model. How would one quantitatively compare these two across all data samples? Recall that in linear regression squared error $\sum(y - y_n')^2$ was used. The binary nature of p_n requires that the error be maximum when the predicted $p(y_n)$ and actual p_n are opposite and vice versa.

Deep Learning
Gradient Descent

In Logistic Regression, an error function or a cost function was generalized by taking a log on the terms so that a summation is obtained instead of a product when one needs to compute across all samples. Note that y_n is the calculated value of probability based on the model and can range between [0,1] and p_n is the target which is either 0 or 1. n is the number of samples.

$$J = -\sum_{n=1}^{N} [p_n \log(y_n) + (1 - p_n) \log(1 - y_n)]$$

J is called the cross-entropy cost function. As a cost function, a negative sign is added in front and with the aim of minimizing the value. Thus, the $b's$ need to be found that minimize this function. This has been done before using calculus in the case of linear regression. It is easy to use the chain rule of differentiation to compute the derivative. But as will be seen this will turn out to be a constant and, thus, b cannot be solved for by setting it equal to 0. Instead, once an initial slope is obtained, gradient descent will be used to iteratively find the location where it is minimum.

Deep Learning
Gradient Descent

Note that the cross-entropy cost function is easily expressed in terms of weights b, by substituting:

$$y = \sigma(z) = \frac{1}{(1 + e^{-z})} \qquad where\ \ z = b_0 x_0 + b_1 x_1$$

The weights,b, can now be found by minimizing J, expressed in terms of b by using the chain rule of differentiation and setting this derivative to 0.

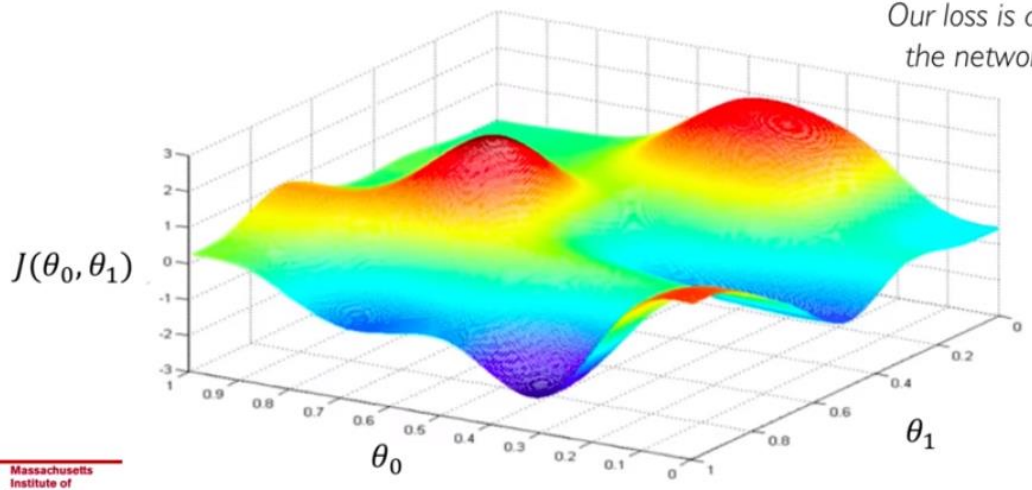
$$\frac{dJ}{db} = 0 \ \Rightarrow \frac{dJ}{dy} * \frac{dy}{dz} * \frac{dz}{db} = 0$$

Deep Learning
Iterative Approach - Gradient Descent

Loss Optimization

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} J(\boldsymbol{\theta})$$

Remember:
Our loss is a function of the network weights!



Deep Learning
Gradient Descent

Rather than setting the above equation to 0, an iterative approach is adopted to solve for the vector b using gradient descent. Start with an initial value for weight vector, b_j, where j is the iteration step and a step size (called the learning rate) and use this slope to iteratively reach the point where J is minimized.

$$b_{j+1} = b_j - Learning\ Rate\ * [p_n - Y_n]^T \circ X$$

A key difference is the way the y and the p are computed. In logistic regression these are evaluated using the sigmoid transformation

$$y = \sigma(b_0 + b_1 x),$$

whereas in linear regression a unit transformation

$$y = b_0 + b_1 x$$

is used, no scaling or transformation is applied to the computed output.

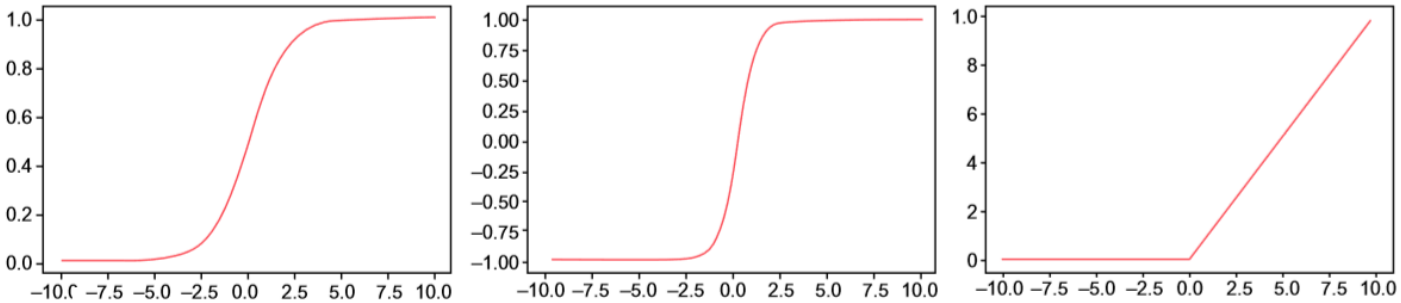
Deep Learning
Activation Functions

This is essentially the concept of an activation function in ANN and deep learning. Think of activation functions like a rule based weighted averaging scheme. If the weighted average $(b_0 + b_1 x)$ crosses a pre-set threshold, the output evaluates to 1, if not it evaluates to 0, which is what happens if the activation function is the sigmoid.

Clearly there are many candidate functions which can be used (sigmoid is simply one of them). Shown below are examples of the most commonly used activation functions used in deep learning:

1. sigmoid,
2. tanh, and
3. rectified linear unit (RELU)

Deep Learning
Activation Functions



FIGURE

Commonly used activation functions: sigmoid (left), tanh (center), and RELU (right). *RELU*, Rectified linear unit.

Observe the similarity in shape between sigmoid and tanh—the only difference between the two is the scaling: tanh scales the output between [-1, 1]. The RELU is also an interesting function—the output linearly increases if the weighted average exceeds a set threshold, otherwise it is set to 0. Note that all three are nonlinear functions which is an important characteristic that enables ANN and deep learning networks to classify linearly non-separable data.

Deep Learning
Activation Functions

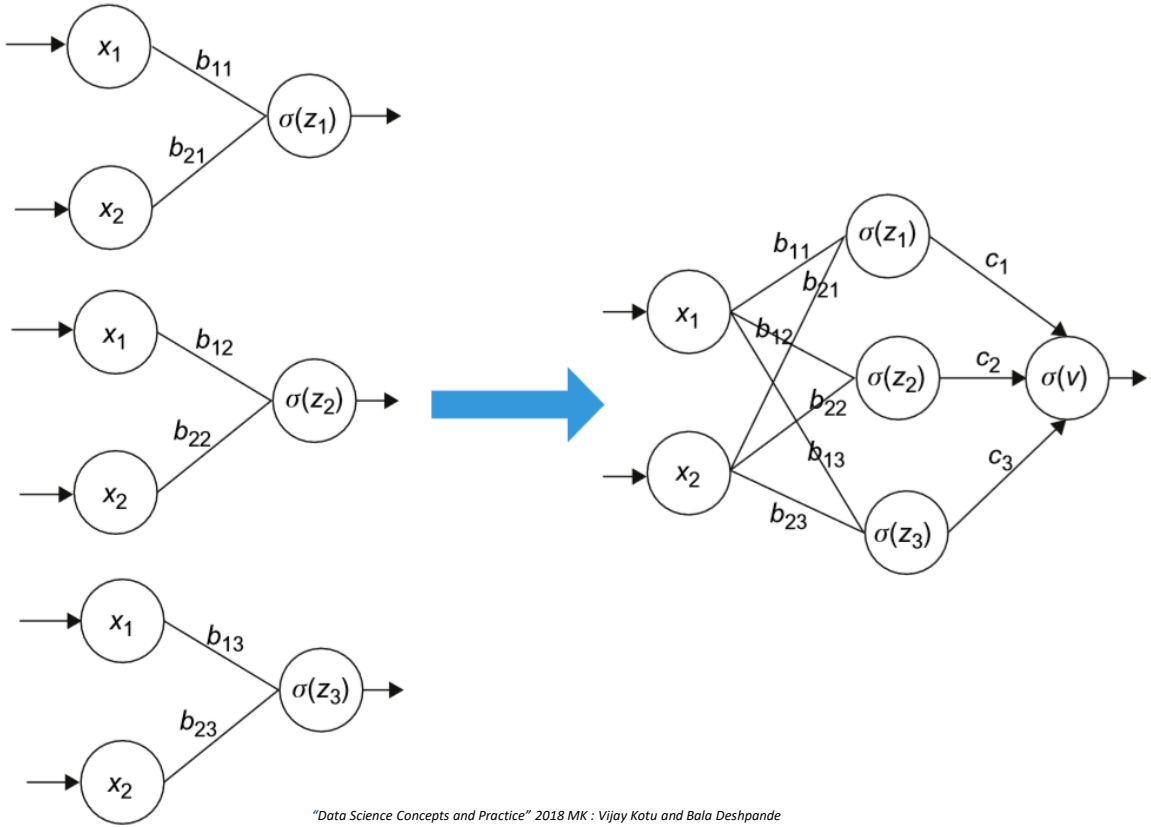
The table below summarizes the formulation of some common quantities that arise in the application of gradient descent for linear and logistic regression. For the cost functions that arise in a general ANN, calculation of the gradient in a closed form, is not feasible, and it has to be computed using numerical approximation. In practice, this is what packages such as TensorFlow (TF) help implement. Such implementations rely heavily on TF for the purposes of gradient calculation and the associated tensor or matrix method bookkeeping that is required.

Method	Cost Function	Derivative of Cost Function	Closed Form Solution	Gradient Form Solution
Linear egression	$J = 1/N \sum_{i=1}^{N} (y_i - b^T x_i)^2$	$dJ/db = 2/N \sum_{i=1}^{N} (y_i - b^T x_i)(-x_i)$	$B = (X^T X)^{-1} Y^T X$	$b_{i+1} = b_i - \eta\, dJ/dB$, where η is the learning rate and $dJ/dB = X^T \bullet (\hat{Y} - Y_i)$
Logistic regression	$J = -\sum (p_n \log y_n + (1 - p_n) \log(1 - y_n))$	$\frac{dJ}{db} = \frac{dJ}{dy}\frac{dy}{dz}\frac{dz}{db}$ where $y = \sigma(z)$ and $z = \Sigma b_i x_i$	None	$b_{i+1} = b_i - \eta dJ/dB$; where $dJ/dB = (P_n - Y_n)^T \cdot X$

Deep Learning
Need for Back Propagation

Weighted averaging is one intuition that would allow a full ANN to be conceptualized starting from the simple two-layer models of logistic and linear regression. Gradient descent is about incrementally updating the weights based on the output generated from a previous iteration. The first iteration is kicked off by randomly choosing the weights. Can the process be made a bit more efficient and "well-rounded" by choosing a series of starting weights in parallel? That is, can one start by building, say 3 logistic (or linear) regression units or models in parallel so that instead of (b_1, b_2) as the starting weights, there are 3 sets: (b_{11}, b_{21}), (b_{12}, b_{22}), (b_{13}, b_{23})? Here the first subscript refers to the input node or feature and the second subscript refers to the node in the intermediate or so-called "hidden" layer. This is illustrated in the next slide. It turns out that by doing this, a small computational price might be paid at each iteration, but the output may be arrived at quicker by reducing the number of iterations. Finally, the output from the hidden layer is once again weight-averaged by 3 more weights (c_1, c_2, c_3) before the final output is computed.

Deep Learning
Need for Back-Propagation



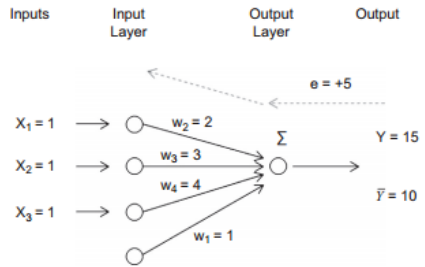
"Data Science Concepts and Practice" 2018 MK : Vijay Kotu and Bala Deshpande

Deep Learning
Need for Back-Propagation

Observe that the right-hand side figure is exactly equivalent to the left-hand side, but representationally simpler. With the hidden layer in place, the output is first computed starting from left to right: that is, $\sigma(z_1)$, $\sigma(z_2)$, $\sigma(z_3)$ are computed as usual and weights for c_1: c_3 are assumed in order to compute the final sigmoid $\sigma(v)$. This would be the first iteration. The output can now be compared to the correct response and the model performance evaluated using the cross-entropy cost function. The goal is now to reduce this error function for which one would first need to incrementally update the weights c_1: c_3 and then work backwards to then update the weights b_{11}: b_{23}. This process is termed "backpropagation" for obvious reasons. Backpropagation remains relevant no matter how many hidden layers or output nodes are in question and is fundamental to understanding of how all ANNs work. The actual mechanics of this computation was described with a simple example in Classification – Neural Networks – next slide.

Deep Learning
Need for Back-Propagation

Recap – Supervised Learning Classification – Neural Networks.



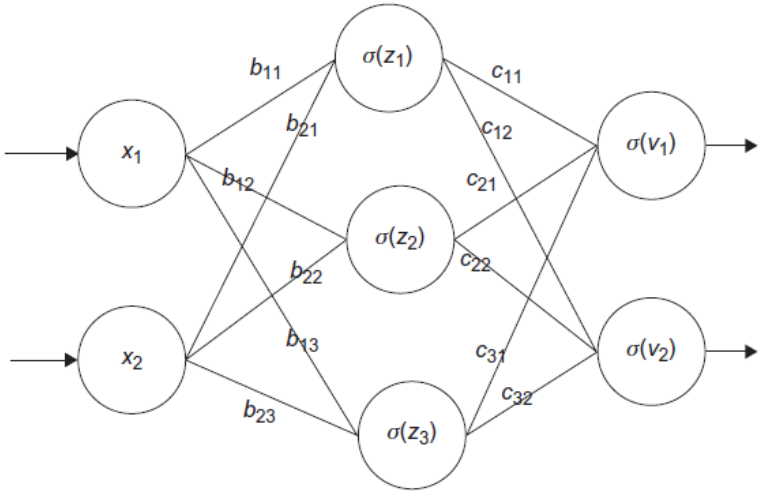
$$w = w' + \lambda * e$$

The main difference between the example above and this present one is the computation of the error function.

Deep Learning
Classifying More than 2 Classes: Softmax

Recall that by using logistic regression one is able to address a binary classification problem. However, most real-world classifications require categorization into one of more than two classes. For example, identifying faces, numerals, or objects, and so on. One needs a handy tool to identify which of the several classes a given sample belongs to. Also recall that in the network models discussed so far in this chapter, there was only one output node and the probability that a given sample belonged to a class was obtained. By extension, one could add an output node for every class that needs to be categorized into and the probability that a sample belonged to that particular class (see next slide) could simply be computed. This is intuitively how the “softmax” function works. Softmax does two calculations: it exponentiates the value received at each node of the output layer and then normalizes this value by the sum of the exponentiated values received at all the output nodes. For instance, when the output layer has two nodes (one each for class one and class two), the probability of each class can be expressed as represented on the next slide.

Deep Learning
Softmax Output Layer in a Network



$$P(Y = Class\ 1): \frac{e^{v1}}{e^{v1} + e^{v2}}$$

$$P(Y = Class\ 2): \frac{e^{v2}}{e^{v1} + e^{v2}}$$

Deep Learning Summary

The triple concepts of activation functions, backpropagation, and (calculus based) gradient descent form what may be loosely termed as the “ABCs” of deep learning and remain at the mathematical core of these algorithms.

There is no strict definition of what constitutes a “deep” learning network. A common understanding is that any network with three or more hidden layers between the input and output layers is considered “deep.” Based on the math described it should be easy to understand how adding more hidden layers increases the number of weight parameters, b_{ij}. In practice it is not uncommon to have networks where the number of weight parameters (termed trainable parameters) run into millions. In the next few sections some typical use cases or applications of deep learning methods will be explored and the practical implementations discussed.

Deep Learning
CNN's and Recurrent Neural Networks

The field of deep learning has exploded in the last decade due to a variety of reasons. CNN (Convolutional Neural Networks) are one representation of a deep neural network architecture.

The other prominent deep learning method in widespread use is called recurrent neural network (RNN). RNNs find application in any situation where the data have a temporal component.

For example:-time series coming from financial data or sensor data, language related data such as analysing the sentiment from a series of words which make up a sentence, entity recognition within a sentence, translating a series of words from one language to another and so on.

Deep Learning
Convolutional Neural Networks

A convolution is a simple mathematical operation between two matrices. Consider a 6X6 matrix A, and a 3X3 matrix B.

$$A = \begin{bmatrix} 10 & 9 & 9 & 8 & 7 & 7 \\ 9 & 8 & 8 & 7 & 6 & 6 \\ 9 & 8 & 8 & 7 & 6 & 6 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

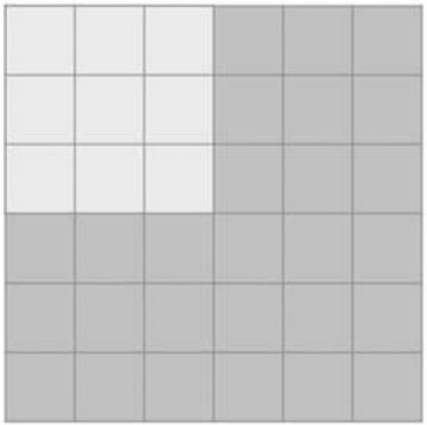
Convolution between A and B, mathematically denoted as A (*) B, results in a new matrix, C whose elements are obtained by a sum-product between the individual elements of A and B. For the given example,

$$C_{11} = 10 * 1 + 9 * 1 + 9 * 1 + 9 * 0 + 8 * 0 + 8 * 0 + 9 * -1 + 8 * -1 + 8 * -1 = 3$$

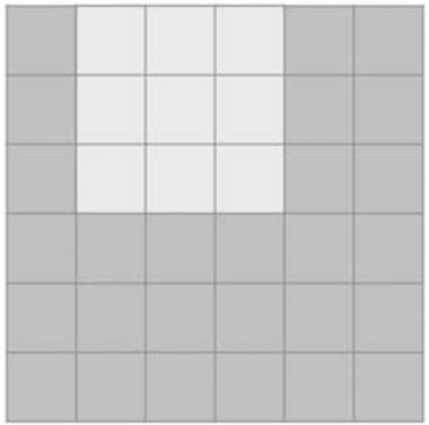
$$C_{12} = 9 * 1 + 9 * 1 + 8 * 1 + 8 * 0 + 8 * 0 + 7 * 0 + 8 * -1 + 8 * -1 + 7 * -1 = 3$$

Deep Learning
Convolutional Neural Networks

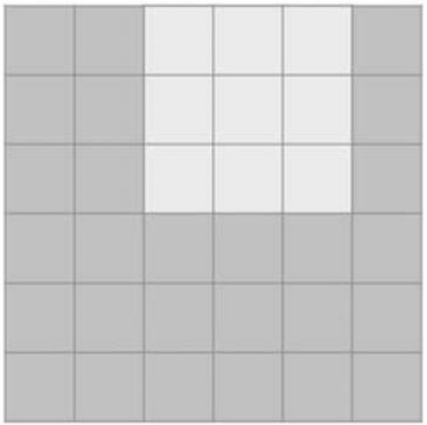
It is easier to understand this operation by visualizing as shown below. Matrix B is the lighter shaded one which essentially slides over the larger matrix A (darker shade) from left (and top) to right (and bottom). At each overlapping position, the corresponding elements of A and B are multiplied and all the products are added as indicated in the figure to obtain the corresponding element for C. The resulting output matrix C will be smaller in dimension than A but larger than B.



$c_{11} = a_{11}b_{11}+a_{12}b_{12}+a_{13}b_{13}+...+a_{33}b_{33}$



$c_{12} = a_{12}b_{11}+a_{13}b_{12}+a_{14}b_{13}+...+a_{34}b_{33}$

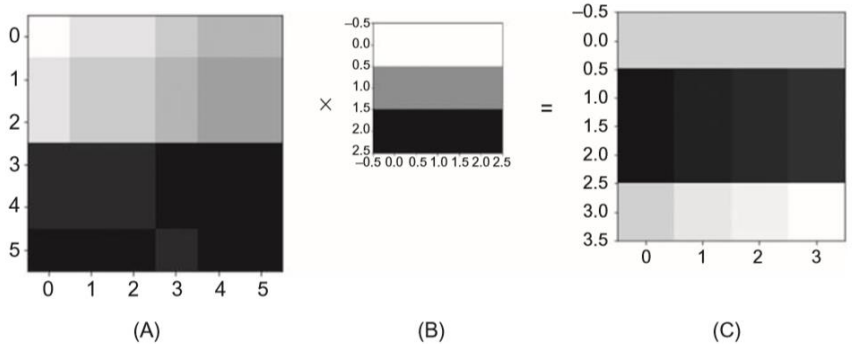


$c_{13} = a_{13}b_{11}+a_{14}b_{12}+a_{15}b_{13}+...+a_{35}b_{33}$

"Data Science Concepts and Practice" 2018 MK : Vijay Kotu and Bala Deshpande

Deep Learning
Convolutional Neural Networks

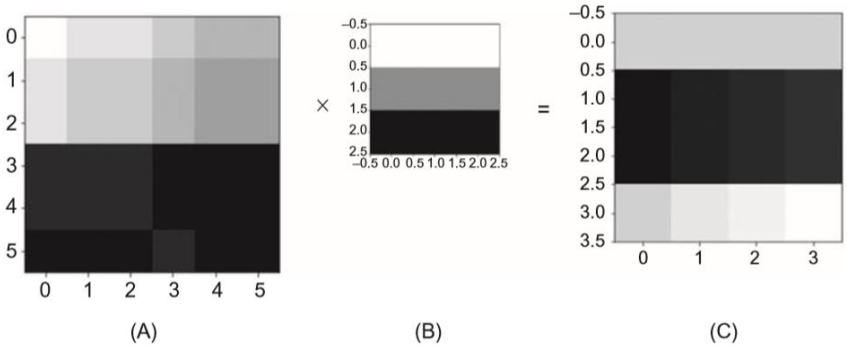
Matrix A is typically a raw image where each cell in the matrix is a pixel value and matrix B is called a filter (or kernel) which when convolved with the raw image results in a new image that highlights only certain features of the raw image. As shown below, A and B are basically pixel maps which when convolved together yield another pixel map, C. One can see that C accentuates or highlights the horizontal edge in A where the pixels jump from a high value to a low value. In this case, the accent appears to be thick—but in the case of a real image where the matrix sizes are in the order of 1000s (e.g., a 1-megapixel image is approximately a 1000 X 1000 matrix of pixels)—the accent line will be finer and clearly demarcate or detect any horizontal edge in the picture.



Deep Learning
Convolutional Neural Networks

Another thing to note about the filter is that when it is flipped by 90 degrees or transposed, that is, use B^T instead of B, a convolution performed on a raw image would be able to detect vertical edges.

By judiciously choosing B, it will be possible to identify edges in any orientation.



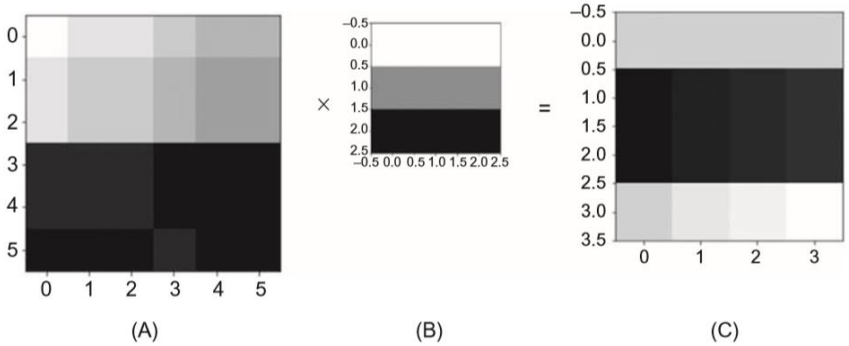
Thus, convolutions are useful in order to identify and detect basic features in images such as edges. The challenge is of course to determine the right filter for a given image. In the example this was done intuitively—however, machine learning can be used to optimally determine the filter values. Observe that determining the filter was a matter of finding the 3 X 3=9 values for the matrix B in the example.

A couple of things to note: the matrix A is $n_{width} * n_{height}$ in terms of pixels for a grayscale image. Standard colour images have three channels: red, green, and blue. Colour images are easily handled by considering a 3-dimensional matrix $n_{width} * n_{height} * n_{channels}$ which are convolved with as many different filters are there are colour channels. one can observe that C is smaller than the raw image A. If the dimensions of A and B are known, the dimensions of C can be determined. For simplicity, assume that $n_{width} * n_{height} = n$ which is quite common in real applications. If the filter is also a square matrix of size f, then the output C is square of dimension $n - f + 1$. In this case $n = 6$ and $f = 3$, therefore, C was 4 x 4.

Deep Learning
Convolutional Neural Networks

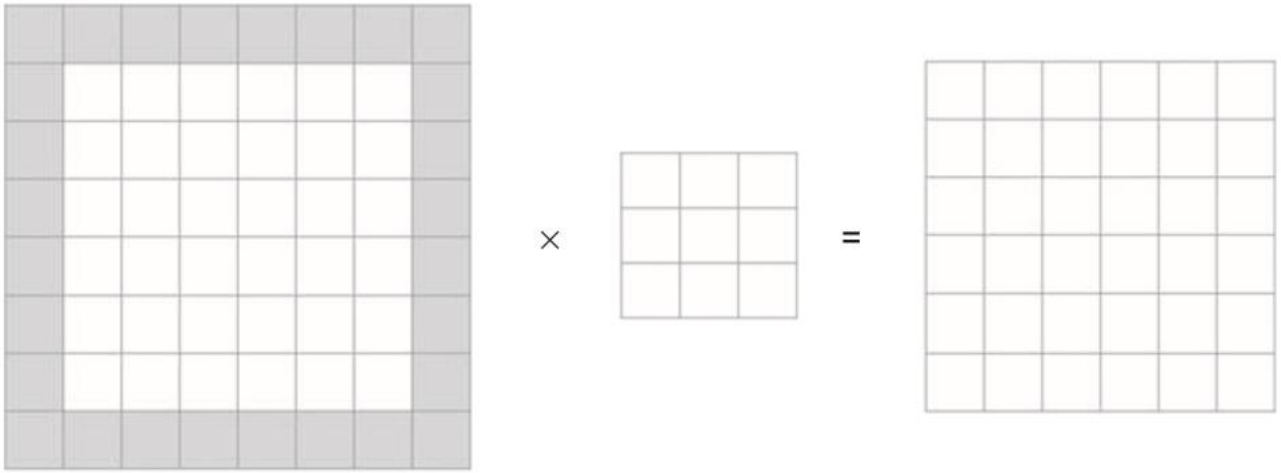
In the diagram one can observe that C is smaller than the raw image A. If the dimensions of A and B are known, the dimensions of C can be determined. For simplicity, assume that $n_{width} * n_{height} = n$ which is quite common in real applications.

If the filter is also a square matrix of size f, then the output C is square of dimension $n - f + 1$. In this case $n = 6$ and $f = 3$, therefore, C was 4 x 4.



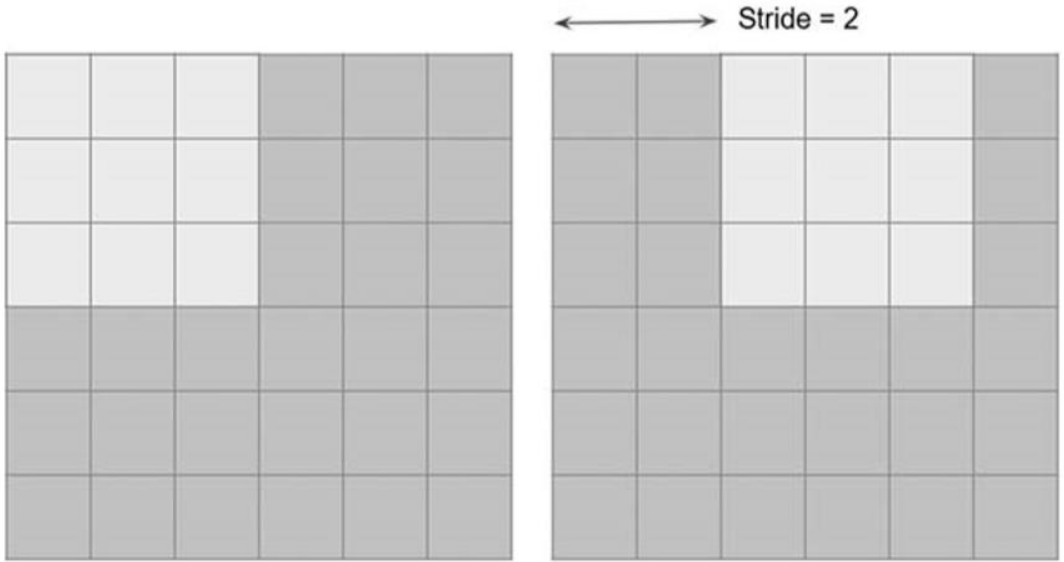
Deep Learning
Convolutional Neural Networks

As the process of convolution reduces the raw image size, it is sometimes useful to enlarge the raw image with dummy pixels so that the original size is retained. This process is called “padding” as illustrated below, If p is the number of padded pixels, then the output dimension is given by $n + 2 * p - f + 1$. Thus, in the example, the output of C will be 6 X 6 if one unit of padding is added.



Deep Learning
Convolutional Neural Networks

Another important consideration in computing convolutions is the “stride.” This is the number of pixels the filter is advanced each time to compute the next element of the output. With stride, s, the output dimension can be computed as $\frac{n+2p-f}{s} + 1$.



Deep Learning
Convolutional Neural Networks

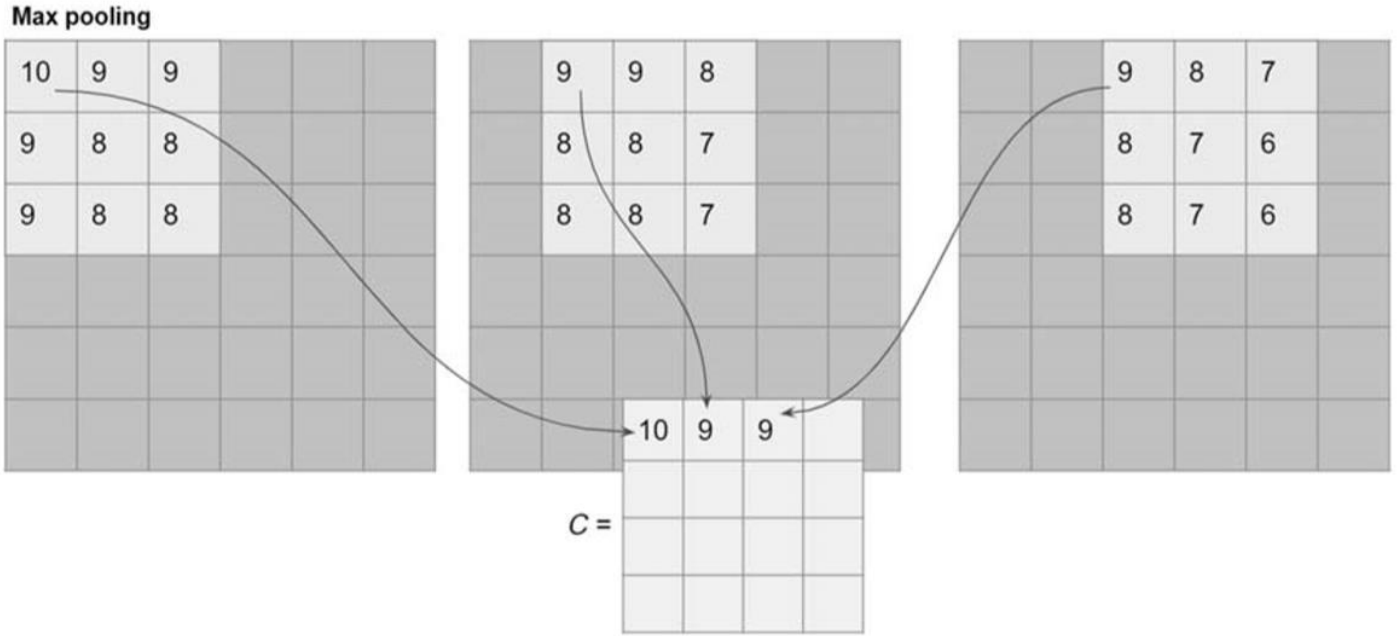
So far, it has been taken on intuition that a convolution helps to identify basic image features such as edges. Thus, a filter strides over an image and at each location of the output a sum-product of the corresponding elements can be computed between the image and filter.

However, instead of performing a sum-product if one simply used the highest pixel value in the overlapping cells at each compute location, a process called max pooling would be obtained. Max pooling can be thought of as an operation that highlights the most dominant feature in an image (e.g., an eye within a face). Max pooling is another feature detection tactic that is widely used for image processing.

Similar to max pooling, an average pooling can also be done to somewhat "airbrush" the raw image, where the values are averaged in the overlapping cells instead of doing a sum-product.

The calculation for the output dimension after pooling still remains $\frac{n+2p-f}{s} + 1$.

Deep Learning
Convolutional Neural Networks – Max Pooling



Deep Learning
Convolutional Neural Networks

A convolution is a linear function which is a general form of any neural network

$Y = f(X)$

$Y = f(g(b))\ \ where\ X = g(b)$

where the weights, B are now the pixels of the filter matrix and the inputs, A are the image pixel values. The sum-product output C is analogous to z in:

$\log\left(\frac{p}{1-p}\right) = b_0 + b_1 x_1$

$and\ p = \left(\frac{1}{1+e^{-z}}\right)\ \ where\ z = b_0 + b_1 x_1$

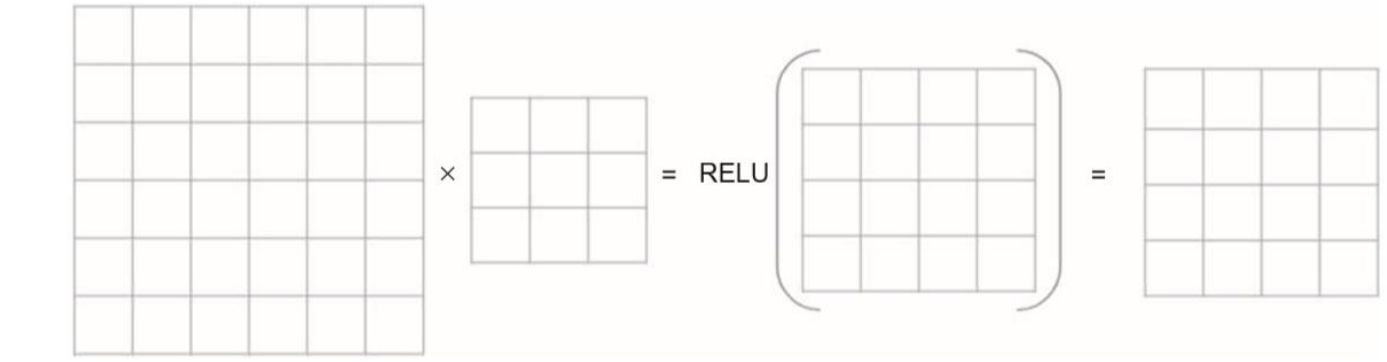
Similar to applying a nonlinear operator to z in

$$p(y) = \sigma(z)$$

the elements of C are typically passed through a RELU nonlinearity.
This entire process, as will be summarized, forms one convolutional layer. The output of this convolutional layer is sent to the next layer which can be another convolutional layer (with a different filter) or
“flattened” and sent to a regular layer of nodes, called a "Fully Connected“ layer.

Deep Learning
Convolutional Neural Networks

Below, A[0] is the raw image and C is the result of its convolution with the filter B. A[1] is the result of adding a bias term to each element of C and passing them through a RELU activation function.



Combining a convolution with an activation function.

Deep Learning
Convolutional Neural Networks

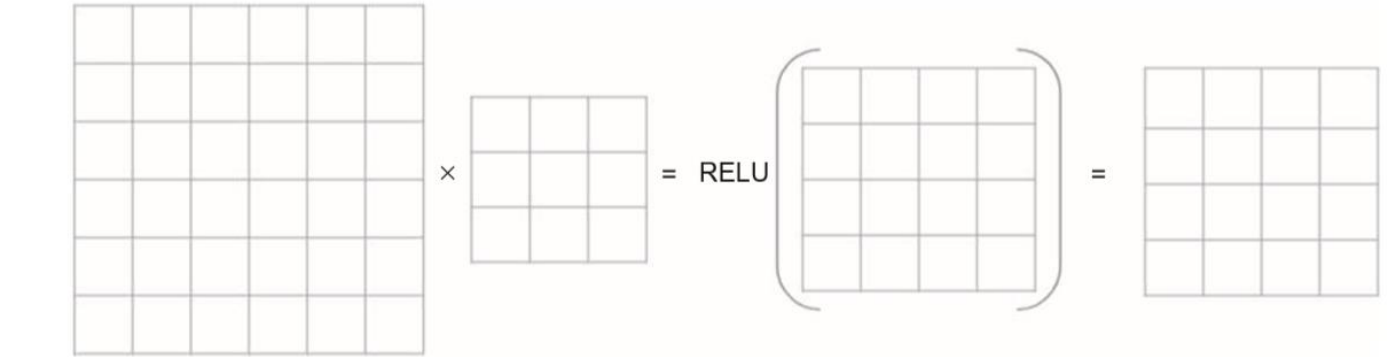
B is analogous to a weight matrix b of

$Y = f(X)$

$Y = f(g(b))$ $where\ X = g(b)$

while C is analogous to σ(z)

$$p(y) = \sigma(z)$$



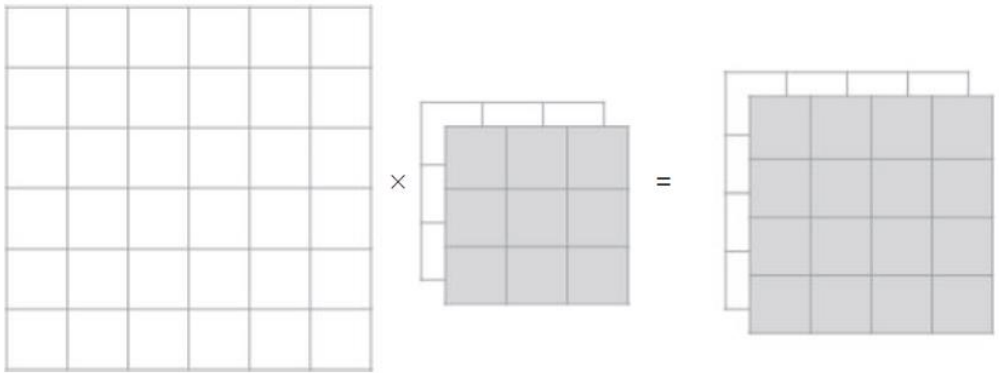
Combining a convolution with an activation function.

Deep Learning
Convolutional Neural Networks

The point of making these comparisons is to highlight how convolutions can be used as a part of a deep learning network. In a neural network, backpropagation can be used to compute the elements of the weight matrix, b, and a similar process can be applied to determine the elements of the filter matrix B.

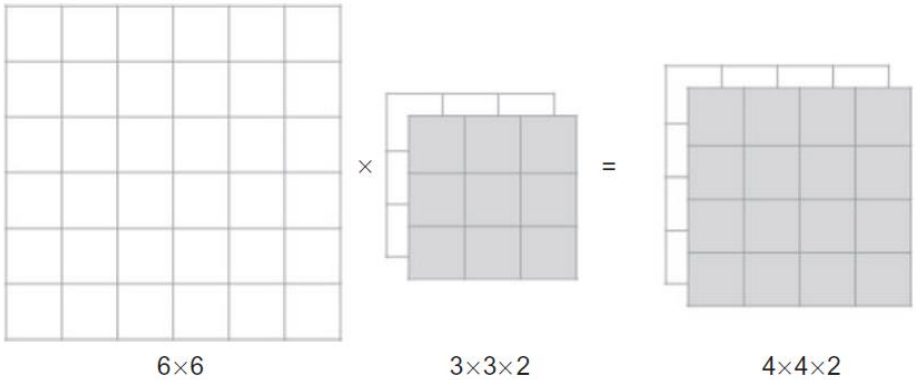
Deep Learning
Convolutional Neural Networks

TensorFlow and such tools allow one to apply multiple filters in the same layer. So for example, one can let B_1 be a horizontal edge detector, B_2 be a vertical edge detector, and apply both filters on A[0]. The output C can be tensorially represented as a volume, in this example, C will have the dimension of 4x4x2, which is essentially a stack of two 4x4 matrices, each one is the result of convolution between A and B_i $i = (1,2)$.



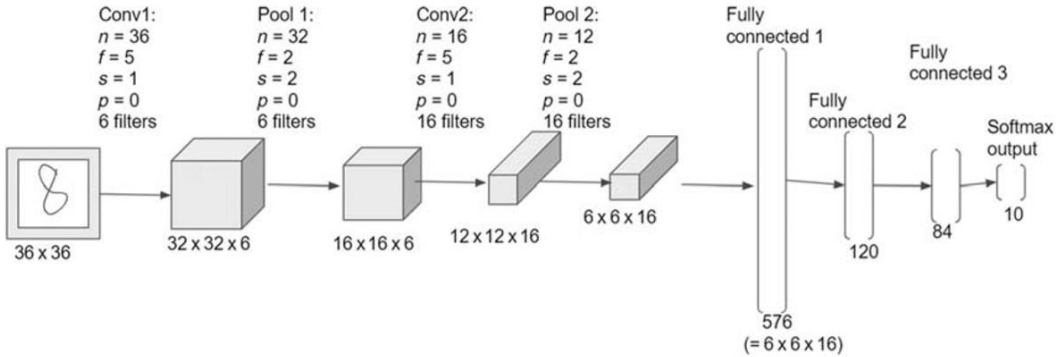
Deep Learning
Convolutional Neural Networks

In order to determine the filter elements, remember that backpropagation will be used. So, a cost-function will need to be computed, and minimized/maximized using gradient descent. The cost function is now dependent on 3X3X2 parameters in this example. In order to completely define the cost-function one can “flatten” the output 4x4x2 matrix into 32(=4x4x2) nodes and connect each node to a logistic regression output node or softmax output nodes.



Deep Learning
Convolutional Neural Networks

Below, a classic CNN architecture introduced by LeCun (1989). It consists of several convolutional layers interspersed with max pool layers and finally followed by what are known as fully connected layers where the last convolution output matrix is flattened into its constituent elements and passed through a few hidden layers before terminating at an output layer. This was one of the first CNNs and was used to recognize handwritten digits.



Architecture of a typical CNN model.

"Data Science Concepts and Practice" 2018 MK : Vijay Kotu and Bala Deshpande

Deep Learning
Convolutional Neural Networks

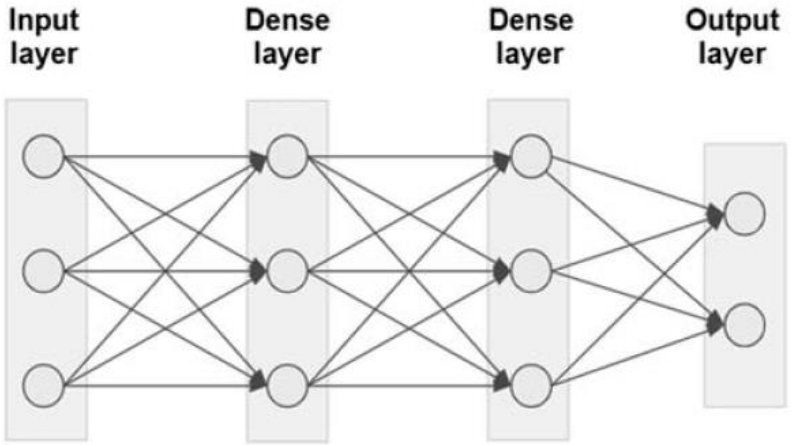
CNNs are highly capable deep learning networks which function highly efficiently because of a couple of reasons. The feature detection layers (such as Conv1, Conv2, etc.,) are computationally quite quick because there are few parameters to train (e.g., each Conv1 filter is 5x5 which yields 25+1 for bias =26 times 6 filters=156 parameters) and not every parameter in the output matrix is connected to every other parameter of the next layer as in a typical neural network (as happens in the fully connected layers down the network). Thus the fully connected layers FC1 and FC2 have 576 times 120 = 69,120 parameters to train. Because of their flexibility and computational efficiency, CNNs are now one of the most common deep learning techniques in practical applications.

Layers are the high-level building blocks in deep learning. There are several convolutional layers and several fully connected layers (also called “Dense” layers). Each layer receives the inputs from the previous layer, applies the weights and aggregates with an activation function. Some additional key concepts that are in usage in deep learning are summarized In the following slides.

Deep Learning
Convolutional Neural Networks

Dense Layer

A dense or fully connected layer is one where all the nodes in the prior layer are connected to all the nodes in the next layer. Several layers of dense layers form different levels of representation in the data.

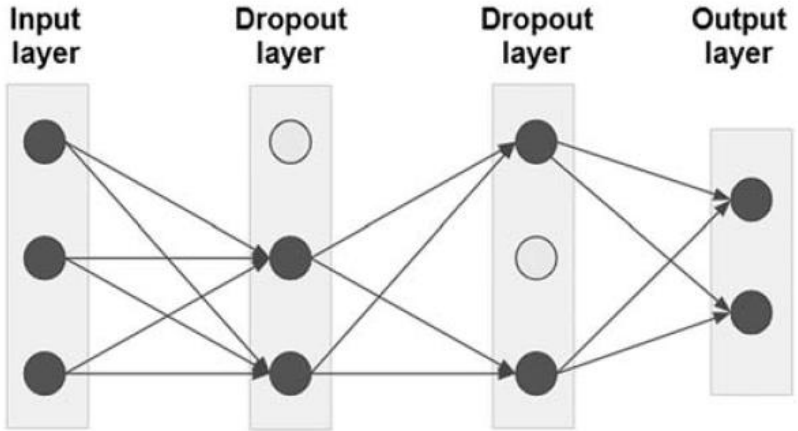


"Data Science Concepts and Practice" 2018 MK : Vijay Kotu and Bala Deshpande

Deep Learning
Convolutional Neural Networks

Dropout Layer

A dropout layer helps to prevent model overfitting by dropping the nodes randomly in the layer. The probability of dropping a node is controlled by a factor which ranges from 0 to 1. A dropout factor closer to one drops more nodes from the layer. This is a form of regularization that reduces the complexity of the model.



"Data Science Concepts and Practice" 2018 MK : Vijay Kotu and Bala Deshpande

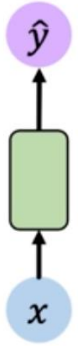
Deep Learning
Recurrent Neural Networks

The field of deep learning has exploded in the last decade. CNN are but one representation of a deep neural network architecture. The other prominent deep learning method in widespread use is the recurrent neural network (RNN).

RNNs find application in any situation where the data have a temporal component. Prominent examples are time series coming from financial data; sensor data such as medical sequences related to EEG's, language related data such as analysing the sentiment from a series of words which make up a sentence, entity recognition within a sentence, translating a series of words from one language to another, interpreting audio as a sequence of sound waves, inferring and understanding genomic sequences and so on.

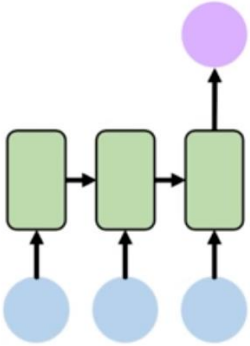
In each of these instances, the core of the network still consists of a processing node coupled with an activation function.

Deep Learning
Traditional Feedforward to Recurrent Neural Networks

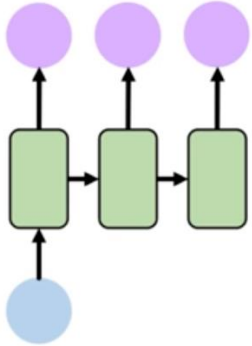


One to One
Binary Classification

"Will I pass this class?"
Student → Pass?

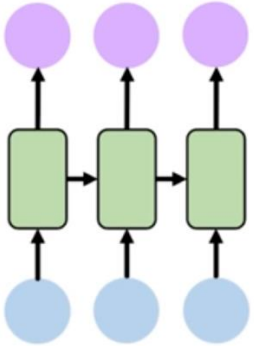


Many to One
Sentiment Classification



One to Many
Image Captioning

"A baseball player throws a ball."



Many to Many
Machine Translation

Deep Learning
Recurrent Neural Networks

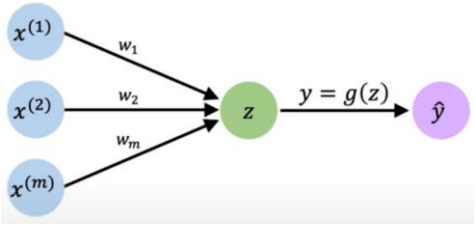
To model sequences we need to be able to: -

- Handle **variable length** sequences
- Track **long term** dependencies
- Maintain information about **order**
- **Share parameters** across the sequence

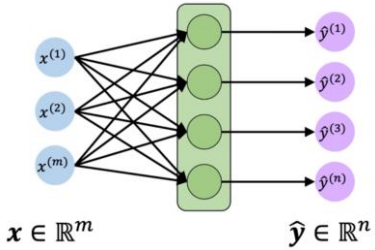
We need to process both a hidden state as well as the current input.

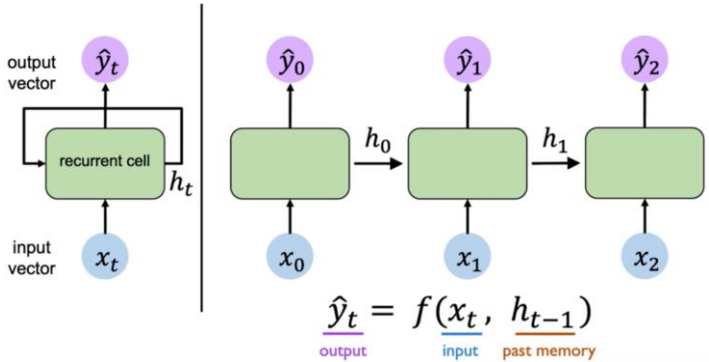
Deep Learning
Recurrent Neural Networks Concepts

Single Timestamp:
Simple Perceptron



With hidden Layer to produce multi-dimensional outputs





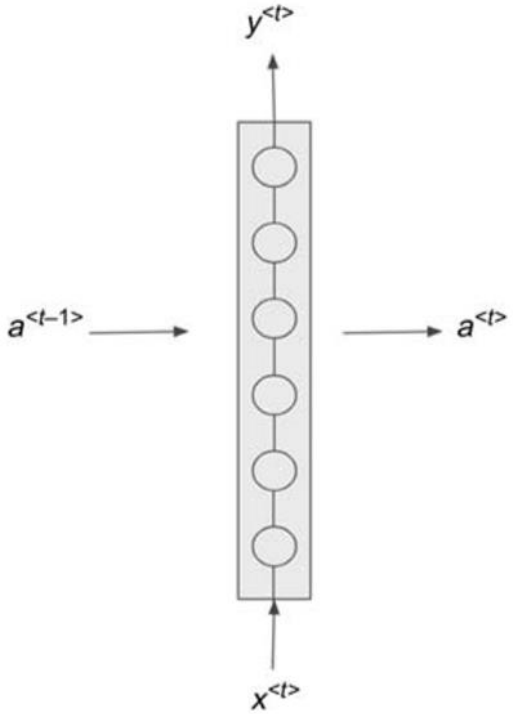
Multiple timestamps:
Output depends on current input

Deep Learning
Recurrent Neural Networks

Suppose the time series problem is a text entity recognition. Below is a set of training examples consisting of sentences from which one can identify certain named entities in each sentence such as proper nouns, places, dates, and so on.

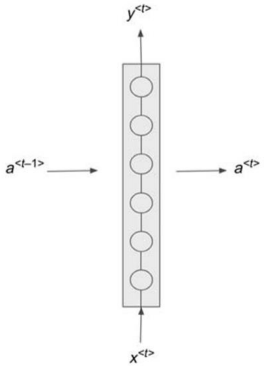
Sample	$x^{<1>}$	$x^{<2>}$	$x^{<3>}$	$x^{<4>}$
1	This	is	an	egg
2	I	love	scrambled	eggs
3	Do	you	like	omelettes?
4	Green	eggs	and	Ham
5	My	name	is	Sam
...	...	...	...	...

Deep Learning
RNN's Basic Representation



"Data Science Concepts and Practice" 2018 MK : Vijay Kotu and Bala Deshpande

Deep Learning
RNN's Basic Representation



The objective here is to predict that $y^{<j>}$ is a named entity such as a proper noun. So, $y^{<4>}$ would be 1, whereas $y^{<1,2,3>}$ are 0's. The idea behind an RNN is to train a network by passing the training data through it in a sequence (where each example is an ordered sequence). In the diagram, $x^{<t>}$ are the inputs where t indicates the position in the sequence. Note that there are as many sequences as there are samples. $y^{<t>}$ are the predictions which are made for each position based on the training data. What does the training accomplish? It will determine the set of weights of this (vertically depicted) network, b_x which in a linear combination with x_i^t and passed through a nonlinear activation (typically tanh), produces an activation matrix $a^{<t>}$

$$a^{<t>} = g(b_x x^{<t>})$$

Deep Learning
Recurrent Neural Networks

RNN's also use the value of the activation from the previous time step (or previous word in the sequence) because typically in most sequences—such as sentences—the prediction of a next word is usually dependent on the previous word or set of words. For example, the previous words "My", "name", and "is" would almost certainly make the next word a proper noun (so $y = 1$). This information is helpful in strengthening the prediction. Therefore, the value of the activation matrix can be modified by adding the previous steps' activation multiplied by another coefficient b_a.

$$a^{<t>} = g(b_a a^{<t-1>} + b_x x^{<t>})$$

Finally, the prediction itself for the position $< t >$ is given by:

$$y^{<t>} = g(b_y a^{<t>})$$

Deep Learning
Recurrent Neural Networks

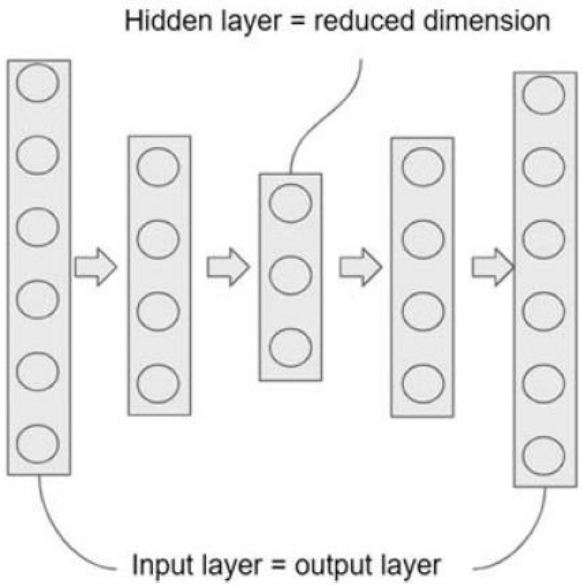
All the coefficients are obtained during the learning process by using the standard process of backpropagation. Because of the temporal nature of data, RNN's typically do not have structures that are as deep as in CNN's. It is not common to see more than 4-5 layers which are all temporally connected.

Deep Learning
Autoencoders

So far, deep learning has been discussed in the context of supervised learning where an explicit labelled output dataset was used to train a model. Deep learning can also be used in an unsupervised context and this is where Autoencoders are useful. An autoencoder is deep learnings answer to dimensionality reduction.

The idea is pretty simple: transform the input through a series of hidden layers but ensure that the final output layer is the same dimension as the input layer. However, the intervening hidden layers have progressively smaller number of nodes (and hence, reduce the dimension of the input matrix). If the output matches or encodes the input closely, then the nodes of the smallest hidden layer can be taken as a valid dimension reduced dataset.

Deep Learning
Autoencoders



"Data Science Concepts and Practice" 2018 MK : Vijay Kotu and Bala Deshpande

Deep Learning
Related AI Models

Two other algorithms will briefly be mentioned which fall more into the domain of AI rather than the straightforward function approximation objective used so far. However, many researchers and experts tend to consider these as newer applications of deep learning because deep networks may be used as part of the algorithms. .

Reinforcement Learning (RL) is an agent-based goal seeking technique where an (AI) agent tries to determine the best action to take in a given environment depending on a reward.

Generative adversarial network (GAN) are at the cutting edge of deep learning implementations —they were first introduced in 2014 and are yet to find mainstream applications. GANs are proposed to be used to generate new samples similar to the data they were originally trained on. For example, creating new “photographs of faces” after being trained on a large set of facial recognition data.

Deep Learning
How to Implement

Deep-learning architecture in RapidMiner can be implemented by a couple of different paths.

The simple artificial neural networks with multiple hidden layers can be implemented by the **Neural Net** operator introduced in Classification, Artificial Neural Network. The operator parameter can be configured to include multiple hidden layers and nodes within each layer. By default, the layer configuration is dense (all the nodes in the prior layer are connected to all the nodes in the next layer). This operator lacks the variety of different layer designs that distinguishes deep-learning architecture from simple Neural Networks.

H2O Deep Learning can be implemented by the Deep Learning operator.

Deep Learning
How to Implement

The Keras extension for RapidMiner offers a set of operators specific for deep learning. It utilizes the Keras neural network library for Python. Keras is designed to run on top of popular deep learning frameworks like TensorFlow and Microsoft Cognitive Toolkit. The Keras extension in RapidMiner enables a top-level, visual, deep-learning process along with data science pre-processing and post-processing.

The Keras extension requires Python and related libraries to be installed. The modelling and execution of the deep-learning process in production application requires machines running GPUs as computation using normal machines can be time consuming.

The following implementation uses Keras extension operators and can be run on general-purpose machines.

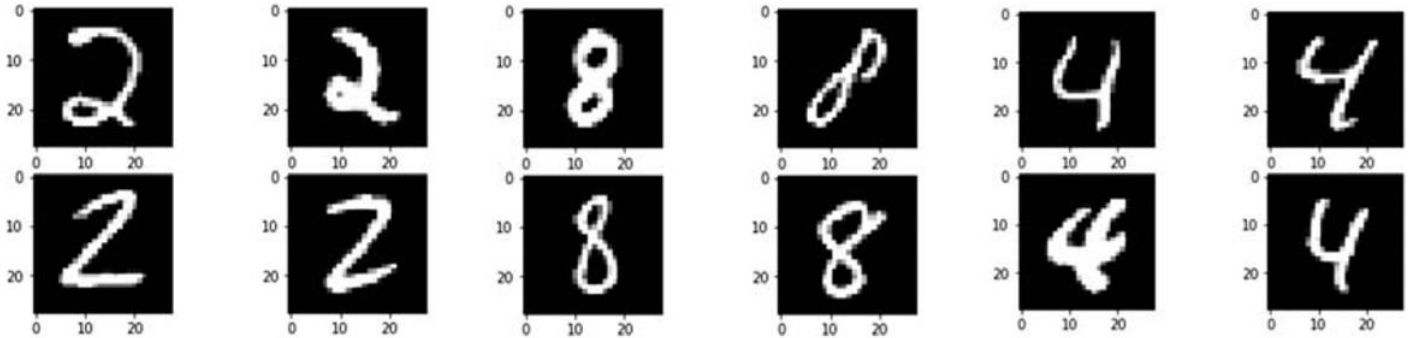
Deep Learning
How to Implement

Handwritten Image Recognition Optical character recognition is an image recognition technique where handwritten or machine-written characters are recognized by computers.

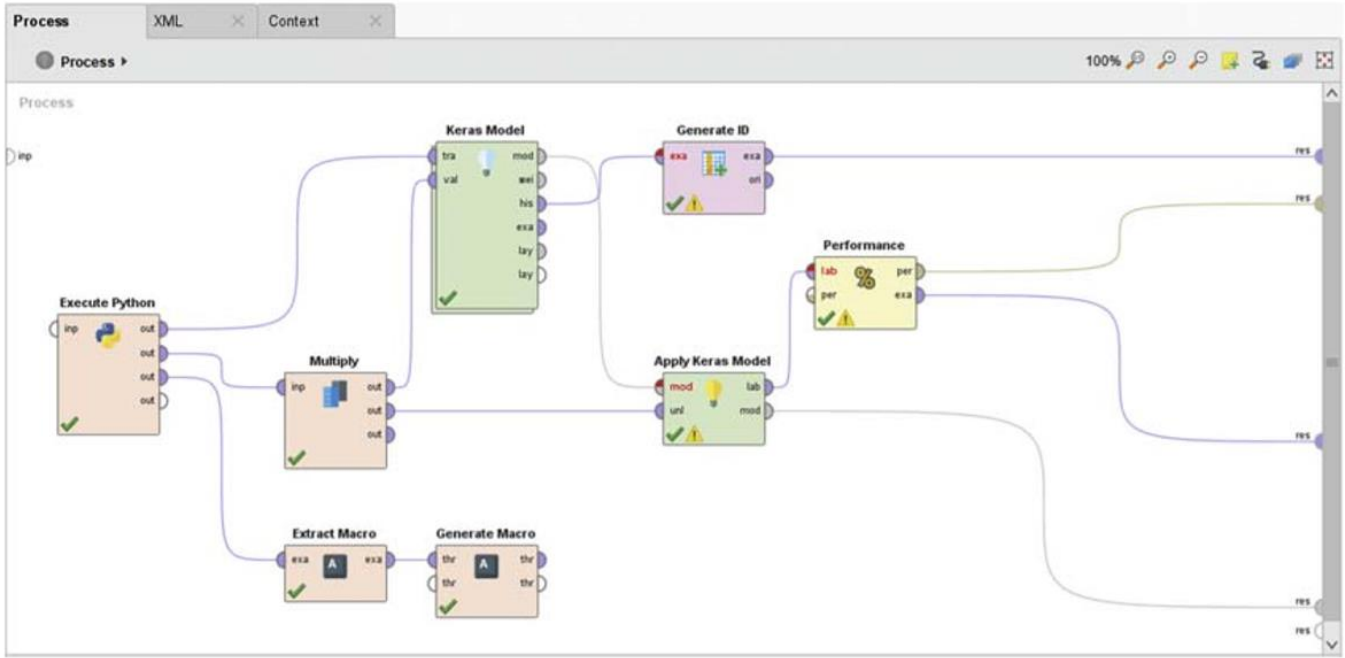
A deep learning-based (convolutional neural network) numeric character recognition model is developed. As with any deep-learning model, the learner needs plenty of training data. In this case, a large number of labelled handwritten images are needed to develop a deep learning model.

Deep Learning
How to Implement

The **MNIST** database (Modified National Institute of Standards and Technology database) is a large database of labelled handwritten digits used commonly for training image processing systems. The dataset consists of 70,000 images of handwritten digits (0,1,2,...,9). Below a sample training images for the digits 2, 8, and 4.



Deep Learning
How to Implement



RapidMiner process for Deep learning

Deep Learning
How to Implement

Step 1: Dataset Preparation

The key challenge in implementing this well-known model using RapidMiner is in preparing the dataset. RapidMiner expects the data in the form of a standard data frame (a table of rows as samples and columns as attributes) organized into rows and columns and in its current version (as of this publication) cannot use the raw image data.

The raw data consist of 70,000 images which are 28 x 28 pixels. This needs to be transformed into a Pandas data frame which is 70,000 rows (or samples) by 784 columns (pixel values) and then split up into 60,000 sample training and 10,000 sample test sets. The 28 x 28 (pixels) yields 784 pixel values.

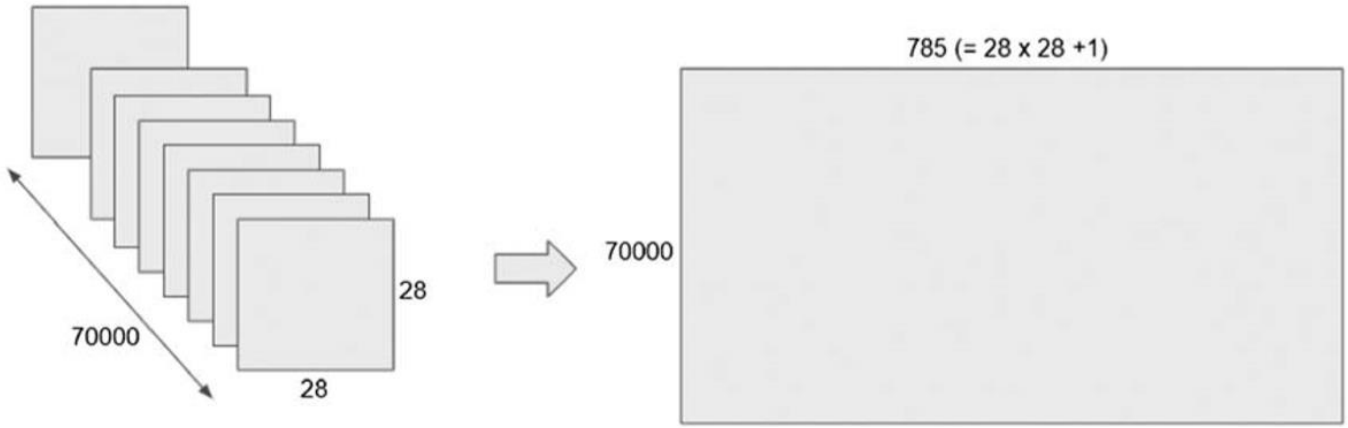
Deep Learning
How to Implement

Step 1: Dataset Preparation (cont.)

The first operator in this process is a Python script executor which takes the raw data and transforms it. The data is read from the MNIST dataset, its shape saved, the pixel data transformed into floats from integers, and finally both training and test “x” and “y” vectors are “unfolded” and merged into Pandas data frames with the “y” column defined as a “label”. The shape information is also returned as a data frame, so that it can later be used to set image sizes and shapes in the convolutional nets. The results from the Execute Python block consists of three data frames: (i) a training data frame 60,000 rows x 785 columns; (ii) test data frame 10,000 x 785; and (iii) and a shape information data frame which stores information about the data (each image is 28 x 28 x 1 tensor, the 1 refers to the channel). After flattening the 28 x 28 image we have 784 columns and we add one more column to contain the “label” information (i.e., which digit is encoded by the 784 columns).

Deep Learning
How to Implement

Step 1: Dataset Preparation (cont.)



Deep Learning
How to Implement

Step 2: Modelling using the Keras Model

The rest of process is very straightforward after the image data is transformed. It passes the training data through the Keras Model and applies the model on the test dataset (Apply Keras Model) and then checks the model performance. The Extract Macro and Generate Macro operators pull the shape information into variables called img_shape, img_row, img_cols, and img_channels which are the used to reshape the data frame into a tensor that the Keras Model operator can then use. The main modelling operator for deep learning is the Keras Model operator under Keras > Modelling folder. The Keras operator is a meta operator which has a subprocess of deep-learning layers within it.

Deep Learning
How to Implement

Step 2: Modelling using the Keras Model (cont.)

The main parameters for the Keras modelling operator chosen are:-

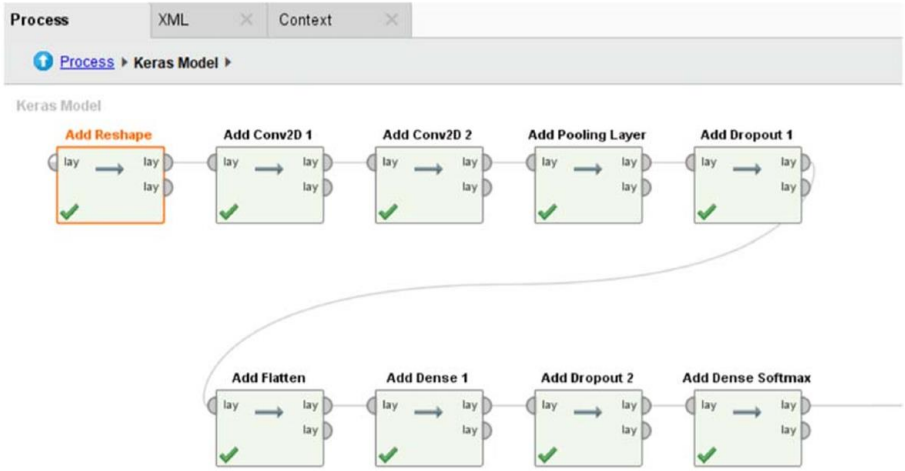
- **Input shape**: [img_size (=img_rows*img_cols*img_channels)],
- **Loss**: categorical_crossentropy,
- **Optimizer**: Adadelta,
- **Learning rate, Epochs, batch_size** initially set at 1.0, 12 and 128 respectively.

These hyper-parameters can be adjusted later for optimal performance.

Deep Learning
How to Implement

Step 2: Modelling using the Keras Model – CNN Architecture (cont.)

There are two Conv2D layers: the first uses 32 filters (n=32) and the second uses 64 filters. All filters are 3 x 3 (i.e., f=3) and stride (s=1). MaxPooling (with f=2 and s=2). A Dropout layer randomly eliminates 25% of the nodes. A flattened layer converts the 2D images into a column of 9216 nodes. This is connected to a 128-node layer, and another Dropout layer (50%) is applied before terminating in a 10-unit output layer.



Deep Learning
How to Implement

Layer (type)	Output Shape	Param #
reshape_1 (Reshape)	(None, 28, 28, 1)	0
conv2d_1 (Conv2D)	(None, 26, 26, 32)	320
conv2d_2 (Conv2D)	(None, 24, 24, 64)	18496
max_pooling2d_1 (MaxPooling2	(None, 12, 12, 64)	0
dropout_1 (Dropout)	(None, 12, 12, 64)	0
flatten_1 (Flatten)	(None, 9216)	0
dense_1 (Dense)	(None, 128)	1179776
dropout_2 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 10)	1290

Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0

A summary of the deep-learning model and its constituent layers.

This model has more than one million weights ("Trainable params") that need to be trained or learned making it a truly deep-learning model. Observe that the largest number of weights occur in the first dense layer.

Deep Learning
How to Implement

Step 3: Applying the Keras Model

The model developed by the Keras modelling operator can be applied on the test dataset using Apply Keras model under Keras > Scoring folder. This operator is similar to other Apply Model operators. The input for the operator is the test dataset with 10,000 labelled image data frame and the model. The output is the scored dataset. The scored output is then connected to the Performance (Classification) operator to compare the deep learning based digits recognition with the actual value of the label.

Deep Learning
How to Implement

Step 4: Results

As seen from the confusion matrix, the performance measured via recall, precision, or overall accuracy is around 99% across all the digits, indicating that the model performs exceptionally well. This implementation shows both the power and the complexity of the deep learning process.

accuracy: 99.04%

	true 7	true 2	true 1	true 0	true 4	true 9	true 5	true 6	true 3	true 8	class precision
pred. 7	1015	4	0	1	0	0	0	0	0	1	99.41%
pred. 2	5	1023	1	0	0	0	0	0	2	2	99.03%
pred. 1	2	1	1130	0	0	1	0	2	0	0	99.47%
pred. 0	1	1	0	975	0	1	3	4	0	3	98.68%
pred. 4	0	1	0	0	969	3	0	1	0	0	99.49%
pred. 9	2	0	0	0	6	994	0	0	0	4	98.81%
pred. 5	0	0	0	1	0	5	881	1	1	1	98.99%
pred. 6	0	0	2	2	5	0	3	949	0	0	98.75%
pred. 3	2	1	2	0	0	3	5	0	1006	1	98.63%
pred. 8	1	1	0	1	2	2	0	1	1	962	99.07%
class recall	98.74%	99.13%	99.56%	99.49%	98.68%	98.51%	98.77%	99.06%	99.60%	98.77%	

Deep Learning Discussion

Deep learning is a fast-growing field of research and its applications run the gamut of structured and unstructured data (text, voice, images, video, and others). Each day brings new variants of the deep architectures and with it, new applications.

There is a fundamental similarity between deep learning and function fitting methods such as multiple linear regression and logistic regression. Activation, back-propagation and Gradient Descent are core mathematical underpinnings.

Common deep learning techniques include CNN's (convolutional neural networks) and RNN's (recurrent neural networks).

Deep Learning
Discussion points to consider

- Large number of parameters to tune

- often requires huge amount of data

– Is my infrastructure suited for that?
– Is the potential ROI high enough for the resource investment?

- For many general ML tasks only minor improvement

- Often not that flexible and very domain specific

- Hard to understand reasoning
  - (consider LIME https://towardsdatascience.com/understanding-model-predictions-with-lime-a582fdff3a3b)

- General Data Protection Regulation (25th of May 2018, EU-based)

– “right to an explanation”
– “prevent discriminatory effects based on race, opinions, health”

Deep Learning
Training using multiple layers of representation of data

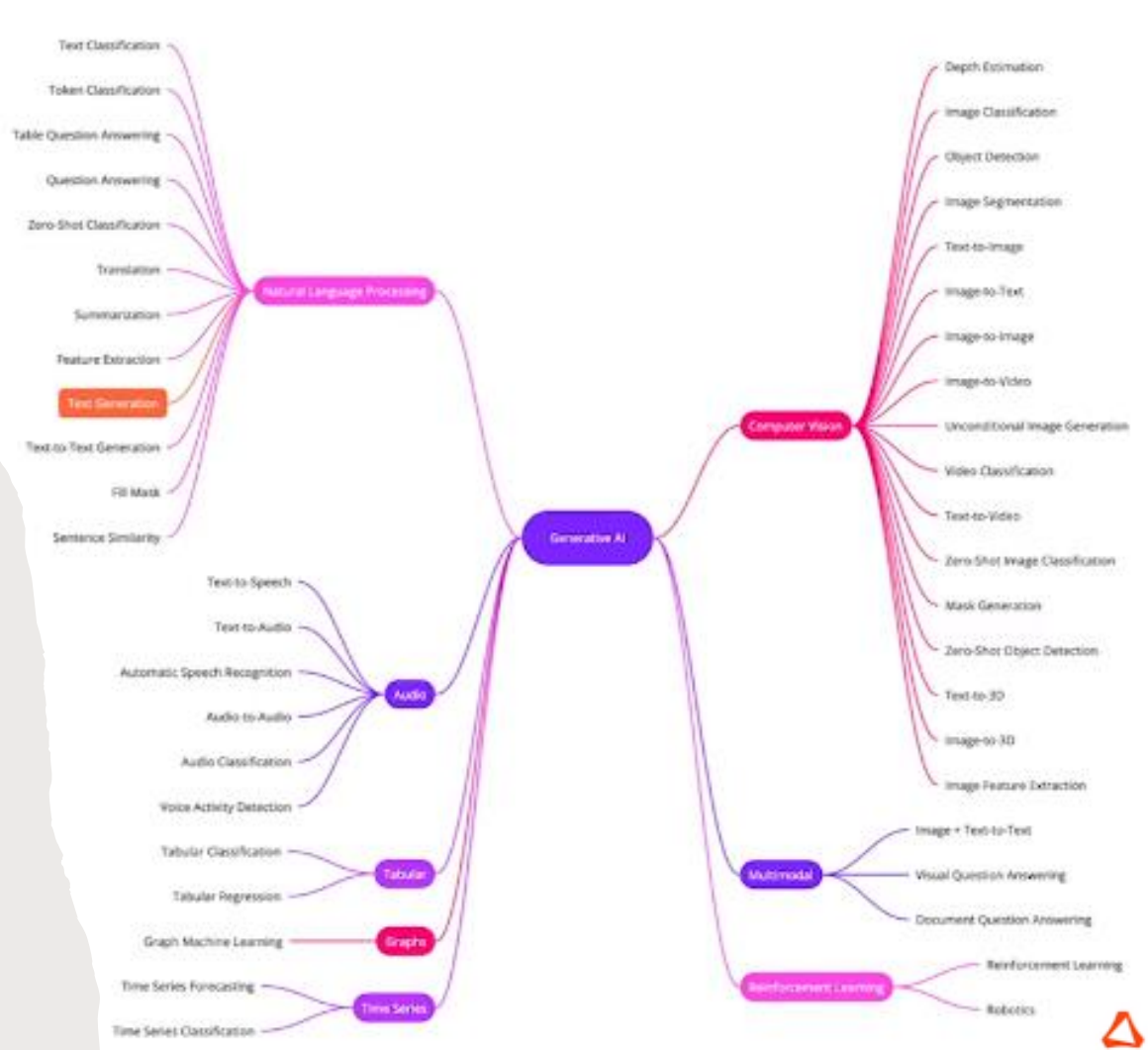
Layer Type	Description	Input	Output	Pros	Cons	Use Cases
Convolutional	Based on the concept of applying filters to incoming two-dimensional representation of data, such as images. Machine learning is used to automatically determine the correct weights for the filters.	A tensor of typically three or more dimensions. Two of the dimensions correspond to the image while a third is sometimes used for color/channel encoding.	Typically the output of convolutional layer is flattened and passed through a dense or fully connected layer which usually terminates in a softmax output layer.	Very powerful and general purpose network. The number of weights to be learned in the conv layer is not very high.	For most practical classification problems, conv layers have to be coupled with dense layers which result in a large number of weights to be trained and thus lose any speed advantages of a pure conv layer.	Classify almost any data where spatial information is highly correlated such as images. Even audio data can be converted into images (using fourier transforms) and classified via conv nets.
Recurrent	Just as conv nets are specialized for analyzing spatially correlated data, recurrent nets are specialized for temporally correlated data: sequences. The data can be sequences of numbers, audio signals, or even images	A sequence of any type (time series, text, speech, etc).	RNNs can process sequences and output other sequences (many to many), or output a fixed tensor (many to one).	Unlike other types of neural networks, RNNs have no restriction that the input shape of the data be of fixed dimension.	RNNs suffer from vanishing (or exploding) gradients when the sequences are very long. RNNs are also not amenable to many stacked layers due to the same reasons.	Forecasting time series, natural language processing situations such as machine translation, image captioning.

"Data Science Concepts and Practice" 2018 MK : Vijay Kotu and Bala Deshpande

Generative AI
Discussion points to consider

- Opening up new Use Cases
- Accelerating existing Use Cases – lowering cost by using knowledge not encoded in your data
- Can be applied now to traditional machine learning Use Cases and is improving

Generative AI Use Cases Beyond Chatbots



ALTAIR

Generative AI
Solving Problems in Analytics and Data Science

- **Data Quality** Use of Generative AI models to identify ad fix data quality problems with access to information from a ‘world model.’

- **Data Engineering** Use of Generative AI models for data enrichment – adding additional information from the world model to existing data sources.

- **Machine Learning** Use of Generative AI models to both open up new use cases as well as solve traditional use cases for which access to data is limited as well as excelling at explaining decisions.

Generative AI
Solving Problems in Analytics and Data Science

Outlier Detection with Generative AI

I will describe below some attributes of a person. I want you to think about this person and assess if this is a plausible group of attributes or not.

Annual Income: USD 450,000

Place of Living: New York City

Car: Porsche

Job Title: CEO

Do you think the attributes for this person are plausible? Deliver a degree between 1 and 10 where 1 means not very plausible and 10 means very plausible.

Just deliver the number for the degree, no prose or additional text.

Explain to me how you got to this degree in a new line.

"This person fits the characteristics commonly associated with a high-earning CEO living in a wealthy area and driving a luxury car."

ALTAIR

Generative AI
Solving Problems in Analytics and Data Science

Outlier Detection with Generative AI

I will describe below some attributes of a person. I want you to think about this person and assess if this is a plausible group of attributes or not.

Annual Income: **USD 45,000**

Place of Living: **Los Angeles**

Car: **Ferrari**

Job Title: **Software Engineer**

Do you think the attributes for this person are plausible? Deliver a degree between 1 and 10 where 1 means not very plausible and 10 means very plausible.

Just deliver the number for the degree, no prose or additional text.

Explain to me how you got to this degree in a new line.

"The annual income of USD 45,000 is relatively low for someone living in Los Angeles, where the cost of living is high. Owning a Ferrari usually requires a much higher income. Additionally, the job title of Software Engineer usually earns a higher salary than USD 45,000."

ALTAIR

Machine Learning with Generative AI

Here is some information about some clients:

Age: 54 | Number of support requests: 103 | Monthly payment: USD 150 | Churn: True

Age: 29 | Number of support requests: 2 | Monthly payment: USD 90 | Churn: False

For this client here, please deliver the value for "Churn" as well as a confidence between 1 and 100 for your choice.

Age: 32 | Number of support requests: 256 | Monthly payment: USD 200

Also explain why you made this choice in a new line.

Churn: **True**
Confidence: 95

"I chose "Churn: True" with a confidence of only 95 because the client's age is relatively young at 32. However, they have a high number of support requests at 256 which suggests they may be encountering issues with the service. Additionally, their monthly payment of USD 200 is higher than average, so if they are not satisfied with the service despite paying more, it is very likely they will churn."

Generative AI
Solving Problems in Analytics and Data Science

Machine Learning with Generative AI

Here is some information about some clients:

Age: 54 | Number of support requests: 103 | Monthly payment: USD 150 | Churn: True

Age: 29 | Number of support requests: 2 | Monthly payment: USD 90 | Churn: False

For this client here, please deliver the value for "Churn" as well as a confidence between 1 and 100 for your choice.

Age: **25** | Number of support requests: **10** | Monthly payment: USD **100**

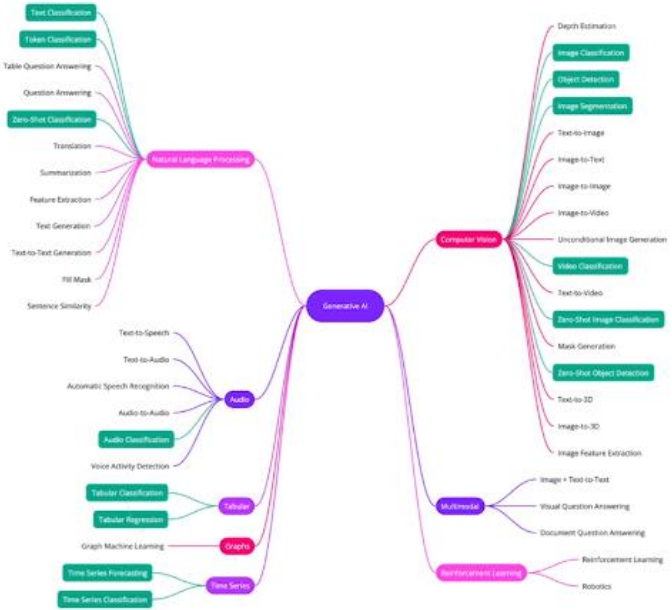
Also explain why you made this choice in a new line.

"Explanation: The second client's details are relatively closer to those of this client. They are both quite young and have a relatively small number of support requests compared to the first client. It is also observed that they pay less monthly payment, and the second client did not churn. However, since we have limited data, the confidence for this prediction is set at 70."

ALTAIR

Generative AI – Hugging Face

Solving Problems in Analytics and Data Science



ALTAIR