Deep learning

by Kevin Kotzé

The application of deep learning methods to particularly challenging statistical learning problems has achieved striking breakthroughs in several fields of study. The renewed interest in deep learning methods were spurred on 2012, when the algorithm was applied to the image net dataset consisting of 1,000 different categories and 1.4 million images to provide impressive results. Handwriting transcription, speech recognition, automatic translation, natural language processing and the current interest in autonomous driving have accelerated the popularity of these methods. Deep learning methods are also used in many aspects of our daily lives and is utilised in most facial recognition software and in the software for various digital assistants. In the coming years, there will be many attempts to apply deep learning to an even wider array of problems.

Deep learning methods also offer a lot of promise for time series forecasting, as well, and may be used to describe temporal dependence and specific features such as trends and seasonality. While the use of deep learning methods may not be the best solution for all time series forecasting problems, they have been successfully applied to problems where classical methods are unable to provide reasonable results, or where other forms of machine learning methods require excessive amounts of feature engineering. In addition, deep learning methods may provide desirable results when working with complex-nonlinear dependencies or a large number of convoluted inputs.

Several researchers have sought to make use of deep learning methods for relatively simple univariate time series forecasting problems, where these models usually provide inferior results to many classical and even naive forecasting methods. This has led to the assertion that they may be unsuitable for time series forecasting purposes. However, more recent research has shown that deep learning methods are effective at more complex time series forecasting problems that involve large amounts of data, multiple potential explanatory variables with complex relationships, and multi-step forecasts or time series classification tasks.

Many applications that make use of deep learning methods for time series forecasting rely on Recurrent Neural Networks (RNNs), where the use of Long Short-Term Memory (LSTM) networks are perhaps the most popular. These methods can work well in some situations, but can often perform better when used in hybrid models with Convolutional Neural Network (CNN) or other neural network structures, where one is able to combine the diverse capabilities of different architectures.

Deep learning may be the future of complex and challenging time series forecasting and this section seeks to provide an introduction to this promising area of research.

1 The evolution and application of deep learning methods

To provide and explanation for the term artificial intelligence (AI) and to show how it relates to both machine learning, and deep learning, we could make use of Figure 1.

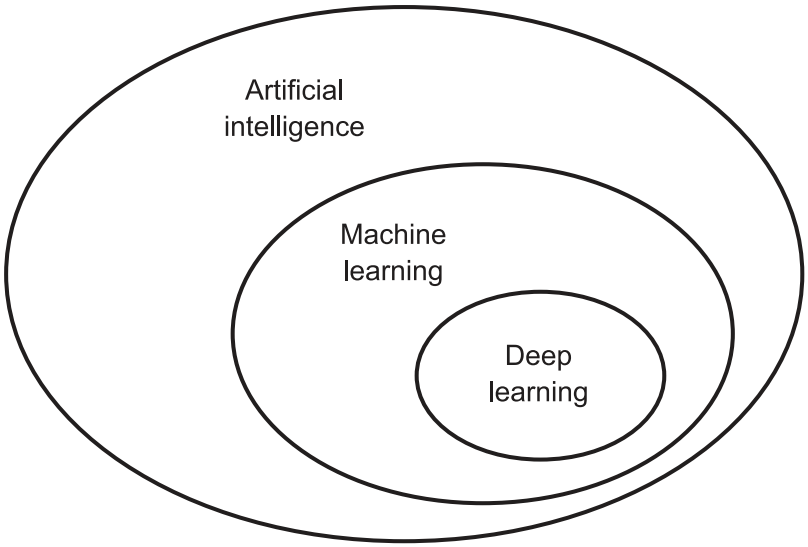

Figure 1: **Artificial intelligence, machine learning & deep learning**

Artificial intelligence was born in the 1950s, when a handful of pioneers from the fledgling field of computer science started asking whether computers could be made to "*think*" - a question whose ramifications we're still exploring today. A concise definition of the field would be as follows: the effort to automate intellectual tasks normally performed by humans.[1] As such, AI is a general field that encompasses machine learning and deep learning, but that also includes many more approaches that don't involve any learning.

Machine-learning systems are usually trained rather than explicitly programmed. For example, we may present a system with many examples relevant to a task, and ask it to find statistical structure in these examples that would eventually allow the system to come up with a set of rules that may be responsible for the data generating process. For instance, if you wished to automate the task of tagging your vacation pictures, you 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. Although machine learning only started to flourish in the 1990s, it has quickly become the most popular and most successful sub-field of AI, a trend driven by the availability of advances in computer hardware and larger datasets.

Deep learning is a specific sub-field of machine learning: a new take on learning representations from data that puts an emphasis on learning successive layers of increasingly meaningful representations. The "*deep*" in deep learning isn't a reference to any kind of deeper understanding achieved by the approach; rather, it stands for this idea of successive layers of representations. 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. Modern deep learning often involves tens or even hundreds of successive layers of representations - and they're all learned automatically from exposure to training data. Meanwhile, other approaches to machine learning tend to focus on learning only one or two layers of representations of the data; hence, they're sometimes called shallow learning.

In deep learning, these layered representations are (almost always) learned via models called neural networks, structured in literal layers stacked on top of each other. The term neural network is a reference to neurobiology, but although some of the central concepts in deep learning were developed in part by drawing inspiration from our understanding of the brain, deep-learning models are not

models of the brain. There's no evidence that the brain implements anything like the learning mechanisms used in modern deep-learning models. You may come across blog articles proclaiming that deep learning works like the brain or was modeled after the brain, but that isn't the case.

1.1 Popularity of deep learning methods

Deep learning methods are responsible for several aspects that influence our daily lives. Between 2010 and 2017, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) was used to evaluate algorithms for object detection and image classification, where the competition made use of a particularly large sample of images. During the 2015 edition of this competition, algorithms were at a point where they could provide a lower error rate than humans. Figure 2 provides the results from the various editions of this competition.

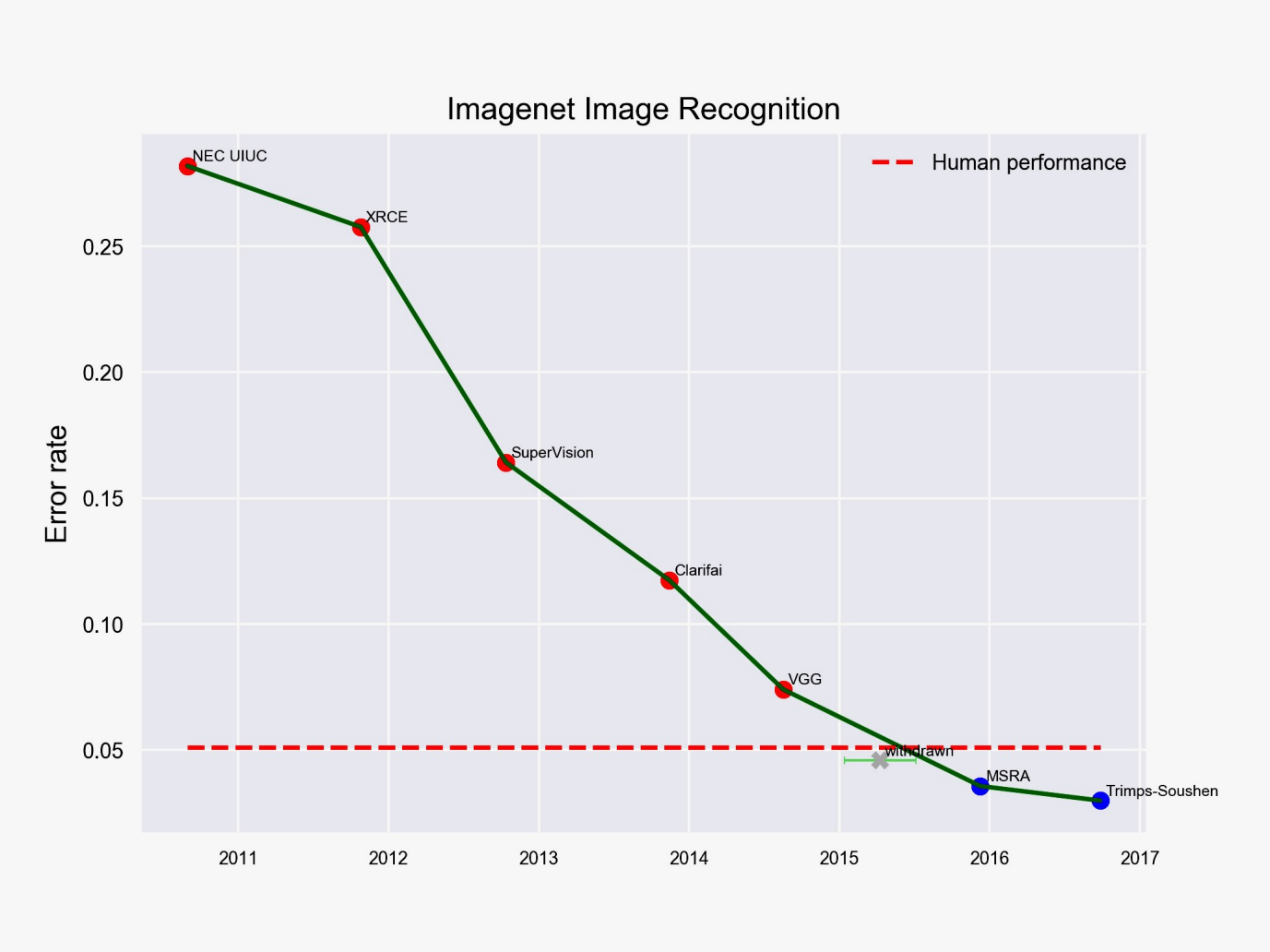

Figure 2: **Image recognition success**

In addition, one of the popular benchmarks within the speech recognition community was also eclipsed at a similar point in time. For example, when using the Hub5'00 switchboard dataset, relatively advanced versions of deep learning algorithms are able to produce a lower error rate than humans. Figure 3 displays the results for selected techniques that have been applied to this dataset.

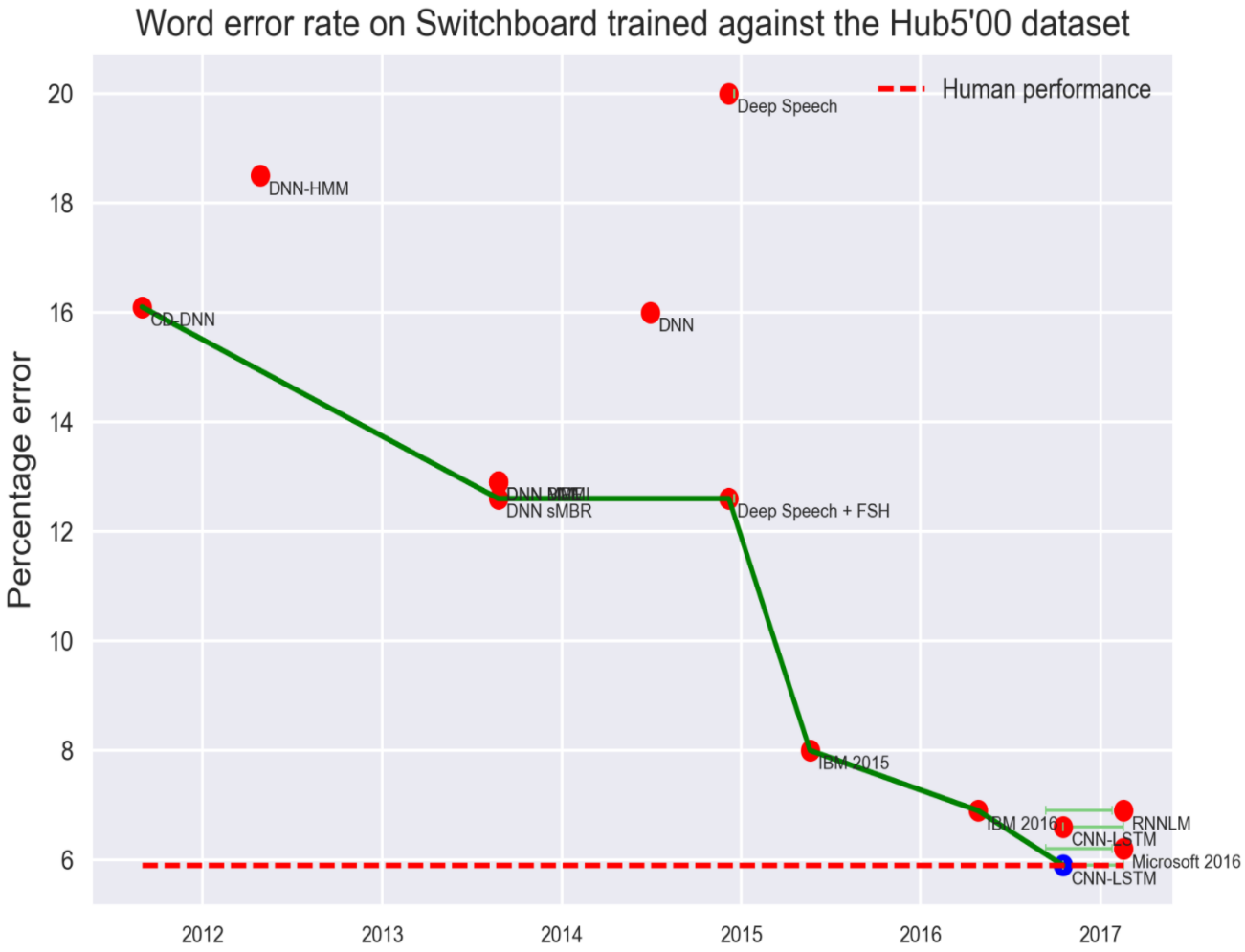

Figure 3: **Voice recognition success**

Other applications that have promoted the use of deep learning methods include its ability to provide near-human-level handwriting transcription, improved text-to-speech conversion, the ability to answer natural-language questions, near-human-level autonomous driving, and improved search results on the web. These methods are also currently used in most digital assistants such as Google Assistant and Amazon Alexa as well as in improved advertising targeting campaigns, as used by Google, Baidu, and Bing.

Although deep learning has led to remarkable achievements in recent years, expectations for what the field will be able to achieve in the next decade tend to run much higher than what will be possible. The risk with high expectations for the short term is that, as technology fails to deliver, research investment will dry up, slowing progress for a long time.

This has happened before. Twice in the past, AI went through a cycle of intense optimism followed by disappointment and skepticism, with a dearth of funding as a result. It started with symbolic AI in the 1960s. In those early days, projections about AI were flying high. One of the best-known pioneers and proponents of the symbolic AI approach was Marvin Minsky, who claimed in 1967, “Within a generation ... the problem of creating ‘artificial intelligence’ will substantially be solved.” Three years later, in 1970, he made a more precisely quantified prediction: “In from three to eight years we will have a machine with the general intelligence of an average human being.” Such an achievement continues to allude us - but in the 1960s and early 1970s, several experts believed it to be right around the corner (as do some people today). One of the unfortunate consequences of setting

extremely high expectations is that researcher and government funding diminished to mark the start of the first AI winter, when progress was relatively lackluster during the 1970s. Figure 4 displays many of the salient breakthroughs that promoted further study in this field.

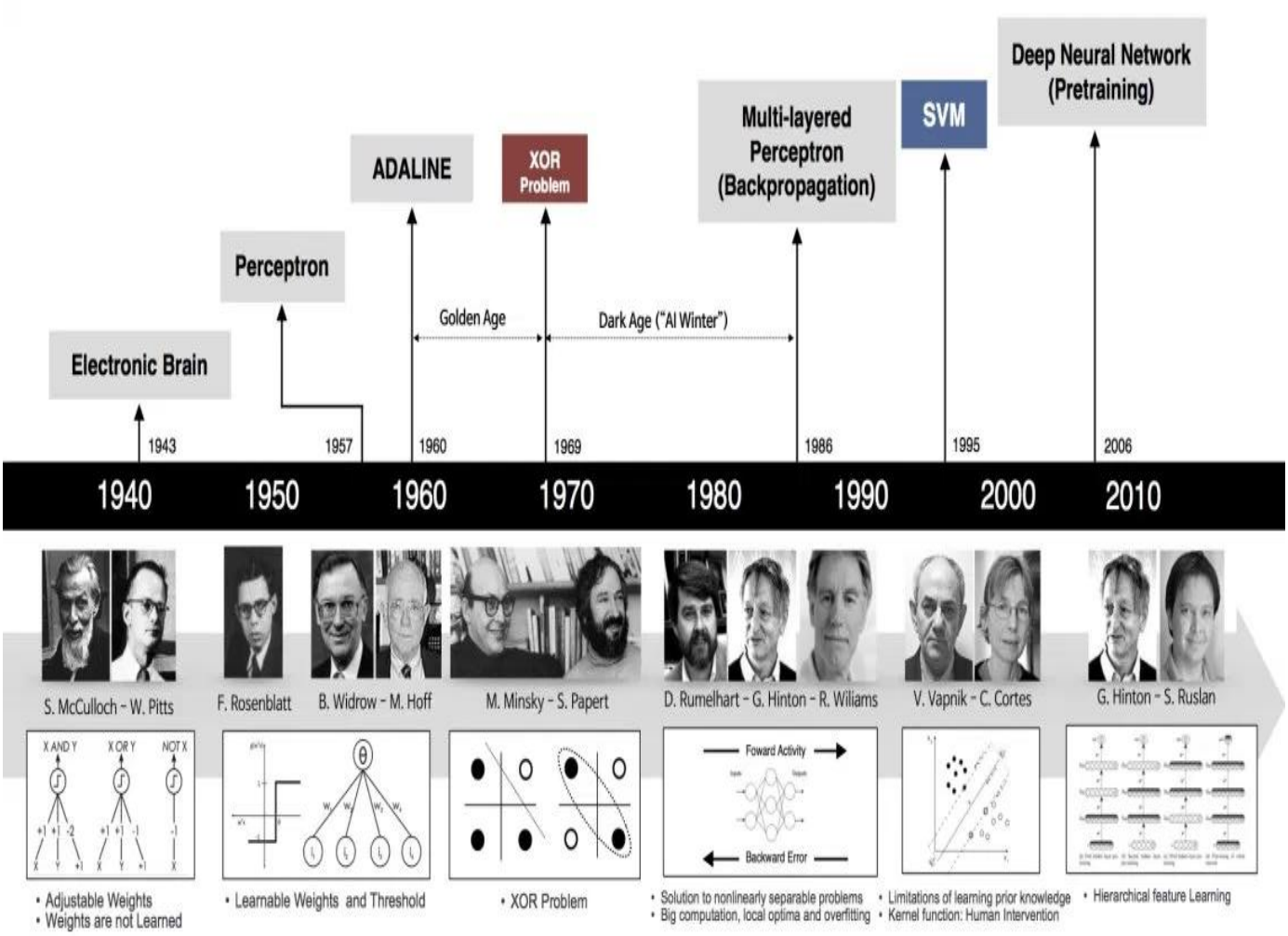

Figure 4: **History of deep learning**

Such an AI winter wouldn't be the last one. In the 1980s, a new take on symbolic AI , expert systems, started gathering momentum among large companies. A few initial success stories triggered a wave of investment, with corporations around the world starting their own in-house AI departments to develop expert systems. Around 1985, companies were spending over $1 billion each year on the technology; but by the early 1990s, these systems had proven expensive to maintain, difficult to scale, and limited in scope. As a result interest in this area of research dissipated, which lead to the second AI winter.

At present, we may be currently witnessing the third cycle of AI hype and disappointment, where we are in the phase of intense optimism. To avoid another potential AI winter it would probably be prudent to maintain moderate expectations about what deep learning can and can't deliver.

2 Statistical inference, machine learning and deep learning

Machine learning involves mapping inputs to outputs by observing many examples of inputs and outputs. Through this process, one is able to gather information and learn about how the use of inputs may produce outputs. Hence, when employing the machine learning methodology, we would use data (i.e. the input) and the output (which could take the form of a forecast) to discover the rules that define the relationship between the input and the output.

This procedure differs to that of traditional statistical inference, that is utilised in classical programming tasks, where we apply the data that we have collected to model that we specified to obtain certain answers. In this sense, the specification of the model would define the rules through which the data is related to specific parameter estimates or forecasts (i.e. the output).

Figure 5 provides a characterisation for how the traditional statistical inference is related to the study of machine learning.

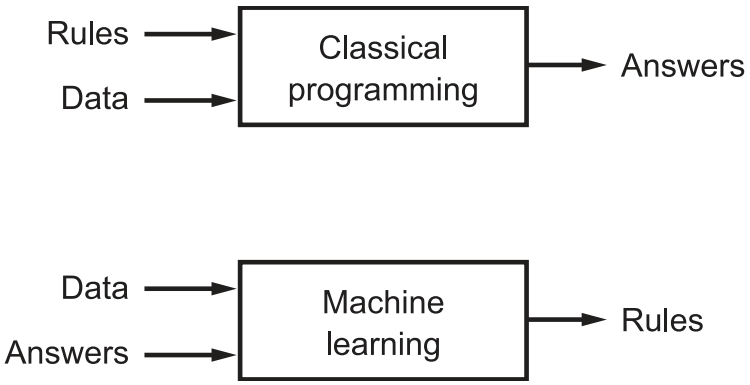

Figure 5: **Similar framework to machine learning**

Deep learning methods make use of layered representations that rely on the application of neural networks models that are structured in layers that are stacked on top of each other to provide a mathematical framework for learning the rules that would allow for the mapping of inputs to outputs. One could consider that in each of the different layers, we perform a simple data transformation, that facilitate an improved (or deeper) understanding of the relationship between outputs and inputs. The specifics regarding the data transformations are learned from the exposure to successive examples. The specification of what a layer does to the input data is stored in the layer weights, where the transformation that is implemented by a layer is parameterised by its weights. Hence, the weights would contain the parameters of a layer. This concept is displayed in figure 6.

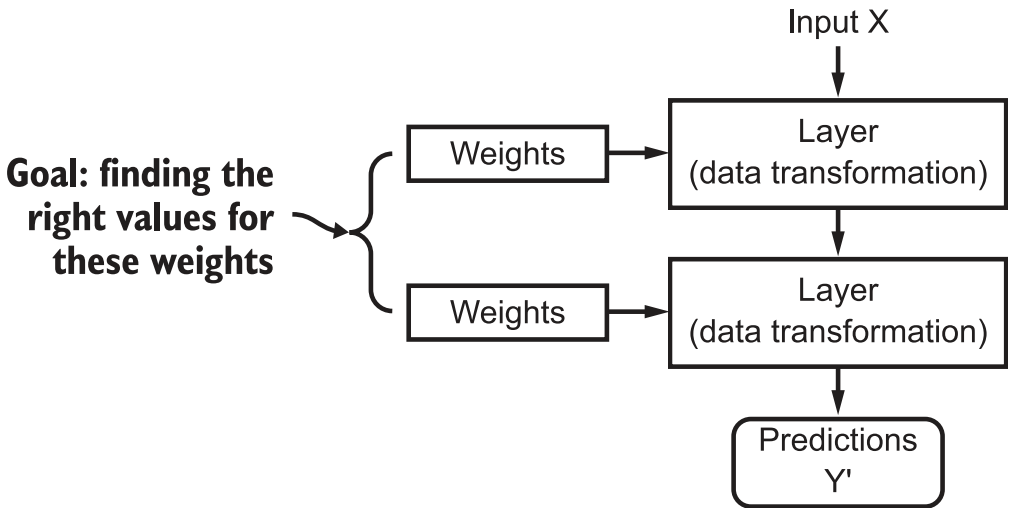

Figure 6: **Model is parameterised by its weights**

3 Training a deep learning model

In this case, the term learning implies finding a set of values for the weights of all the layers in a network, such that the network will correctly map the inputs to their associated output. Note that deep neural networks may contain tens of millions of parameter weights, so finding the correct values for all of them may seem like a daunting task, especially given that modifying the value of one parameter will affect all the others. Hence, the use of these models is computationally expensive and this area of research would not have developed to the extent that it has without the advances in computer hardware.

To determine whether the model has learned something that may be useful from a particular example, we compare the predictions of the network and the true target to compute a distance score. The distance score is then subjected to a loss function that we need to specify. Examples of loss functions include those that were discussed in the section on forecast evaluation (i.e. such as the RMSE). In a deep learning application we would use this score as a feedback signal to adjust the value of the weights a little, in a direction that will hopefully lower the loss score for the subsequent example. This adjustment is the job of the optimiser. Such a workflow is displayed in figure 7.

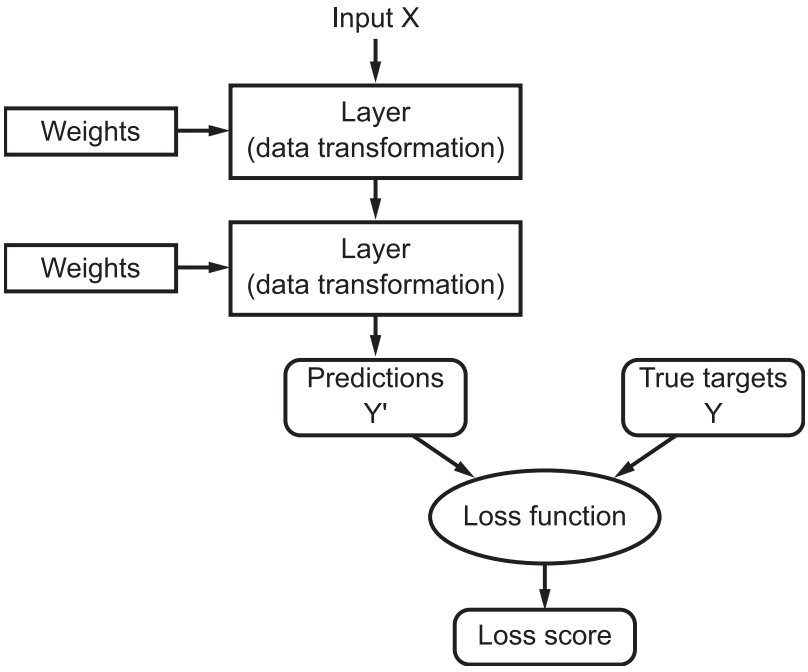

Figure 7: **Loss function measures the quality of output**

For the initial iteration, we may assign random values to the weights of the network, which implies that the model merely implements a series of random transformations. The output from this iteration would usually be relatively different to what the ideal result should be, and the score for the loss function would usually be very high. However, with every example the network processes (through repeated sampling) the weights are adjusted a little in the correct direction, and the score for the loss function will decrease. This is a part of the training procedure, which when repeated a sufficient number of times (typically tens of iterations over thousands of examples), yield values for the weights that minimise the loss function. This looping procedure is illustrated in figure 8.

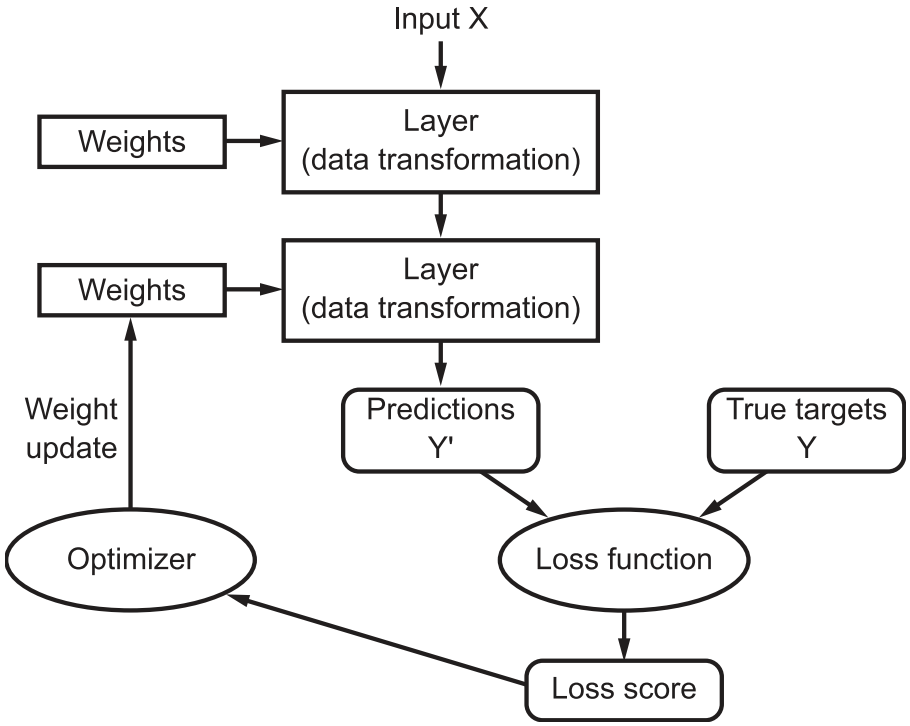

Figure 8: **Loss score used as a feedback to adjust the weights**

To illustrate this concept with the aid of a visual example, consider the representation in figure 9, where we make use of a handwritten number in a transcription classification problem. Such data could be represented by a sequence of binary values that pertain to either of the two pixel colours in the image. For different weights, this data may be transformed in each of the layers to provide different representations of the data.

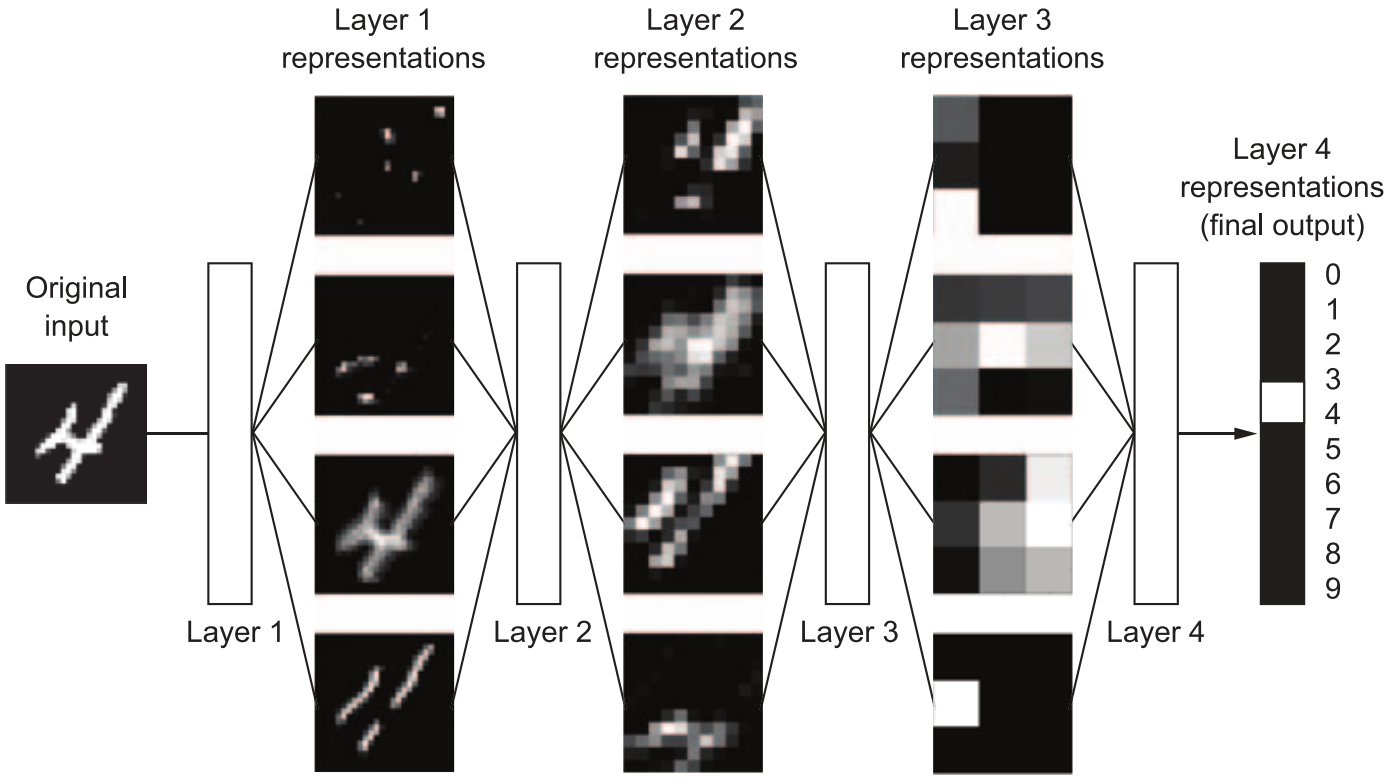

Figure 9: **Neural networks involve several layers**

After training this model on successive examples, where we would use different handwritten samples for a particular number, then we may be able to derive an appropriate general representation at each layer, for what the data should look like for a particular number, which would complete the classification exercise. Hence, one could think a time series problem where we have inputs that take

the form several time series variables. The data would then be subject to different data transformations through the use of different layers, weights and units that are used to describe the different properties of the data. The inputs could then be mapped to the output, which could take the form of a future value for a particular time series, when we are looking to construct a model that could be used for predicting future values of a particular time series. Of course, we could also use such a model for classification purposes by changing the output layer.

3.1 Deep learning layers

Figure 10 provides an example that includes eight different inputs, which could take the form of explanatory time series variables. This particular model structure incorporates three different layers that include nine units each. This model could then be used in either a classification or regression problem to define four outputs. In this diagram we are using feed-forward model, which moves from left-to-right, where each unit in layer one could potentially influence all the other units in layer two, while each of the units in layer two could influence all the other units in layer three.

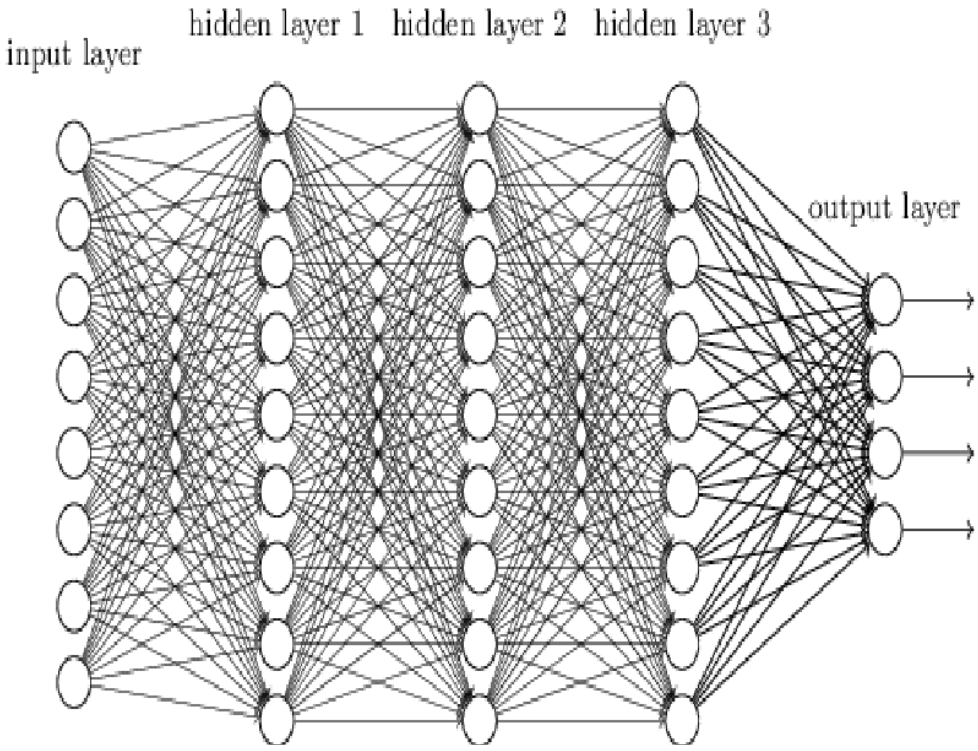

Figure 10: **Feedforward representation**

Alternative modelling architectures allow for additional interactions across layers. For example, in the recurrent neural network (RNN) we allow for the input layer to influence each of the different hidden layers directly, as displayed in figure 11. In addition, this representation would also allow the units within the layers to influence the outputs.

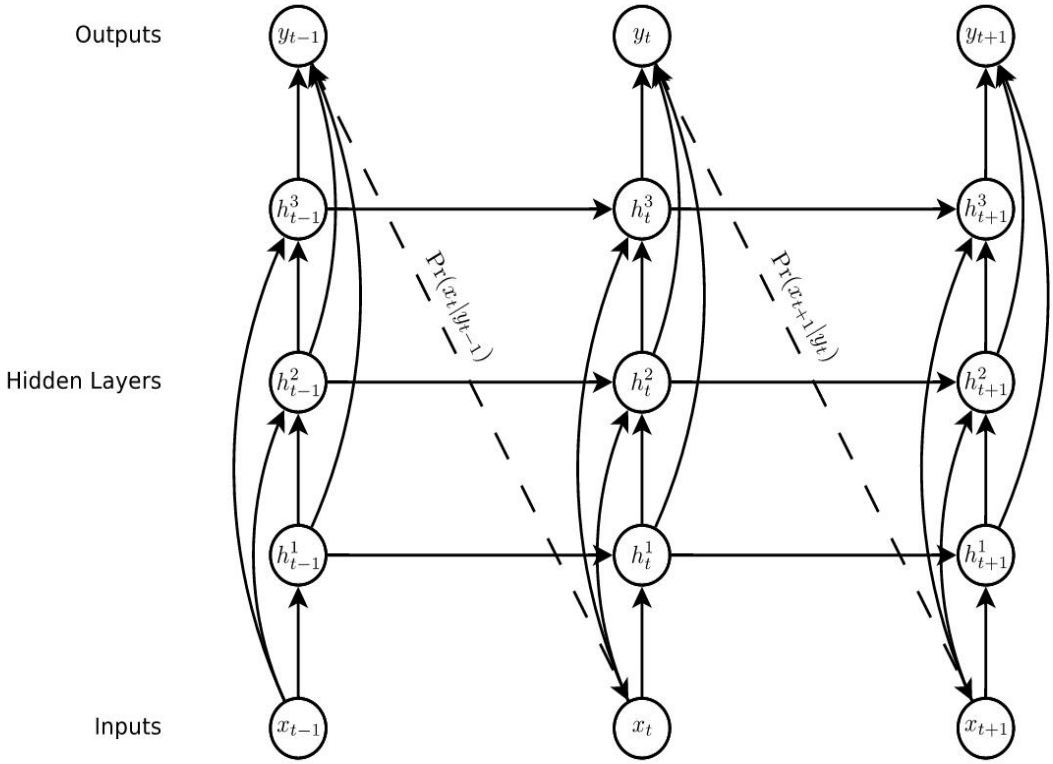

Figure 11: **Recurrent neural network (RNN)**

Given all the potential relationships that could exist in this setup, it would not be surprising to find that some of the potential relationships between the units may be irrelevant. Generating values for the weights that are associated with units that are irrelevant would result in an unnecessary waste of computing time and it may produce a model that is prone to over-fitting errors. To reduce the number of weights that need to be generated we apply a certain degree of dropout, where some of the units are assigned weights of zero when they do not contribute to a reduction in the model score (or loss function). The basic idea behind the concept of dropout is displayed in figure 12.

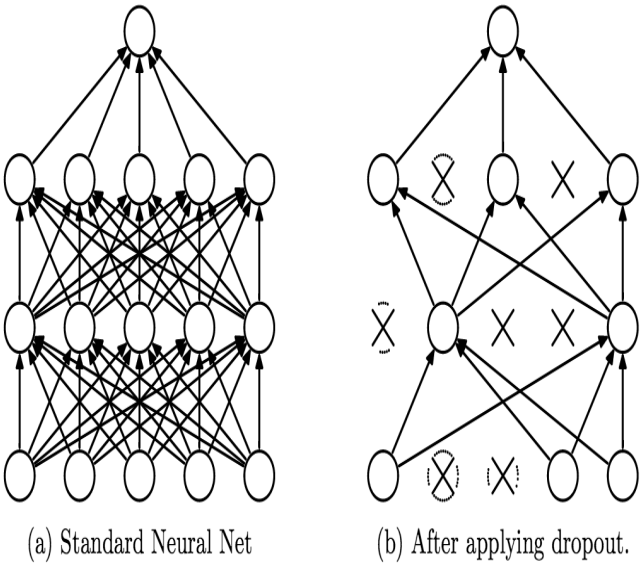

Figure 12: **Dropout**

When applied on mass, the application of the dropout principle could be represented by figure 13.

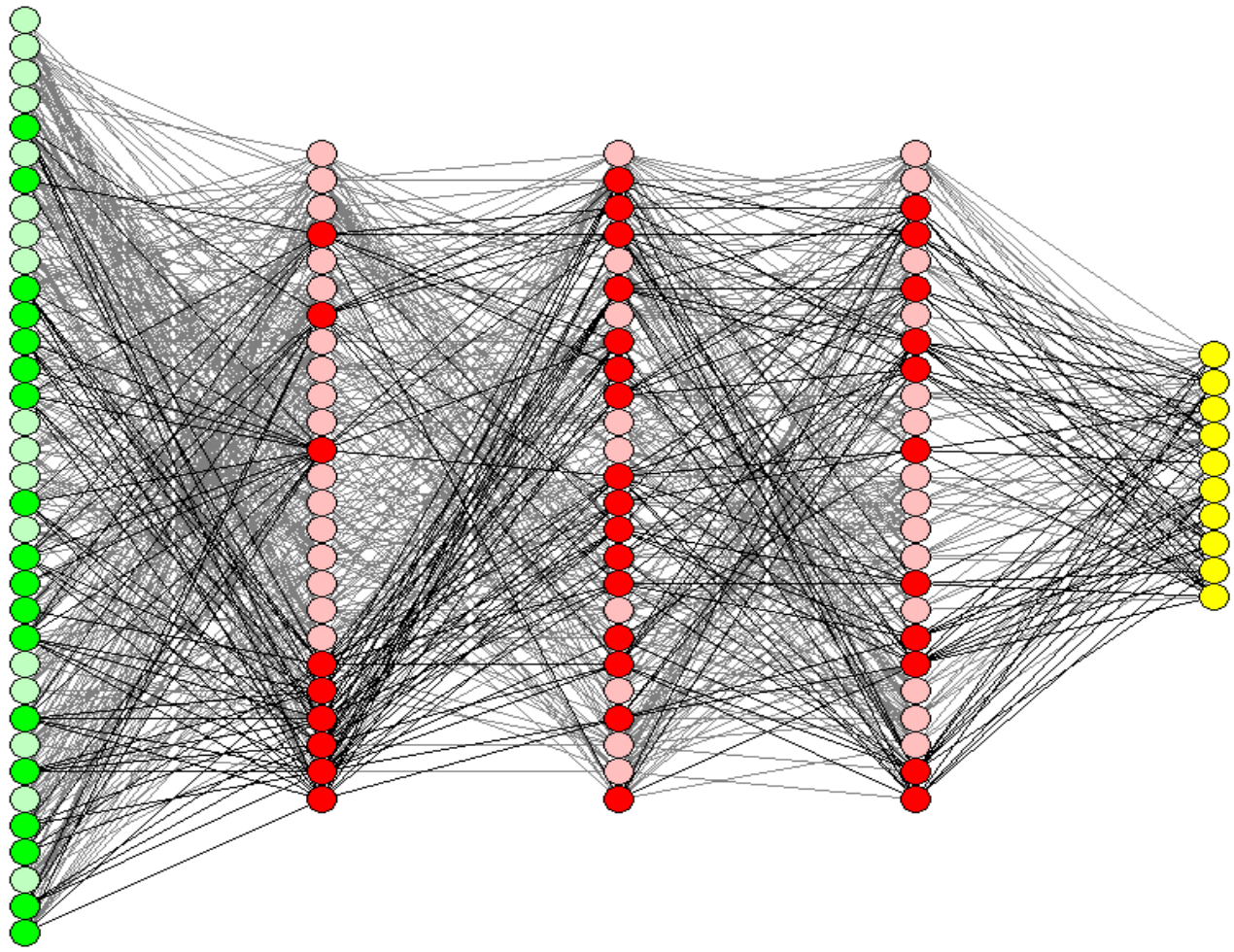

Figure 13: **Dropout on mass**

3.2 Deep learning weights and activation functions

The values for the weights do not share the same features as coefficients in regression models as we are not able to interpret the values for the weights within the hidden layers of a deep learning model. In addition, we are also able to make use of particular transformations of the weights to ensure that the values are bound between specific values. For example, if we were to bound the values between zero and one then we could make use of a sigmoid function, which "squashes" arbitrary values into the $[0, 1]$ interval.

Within the field of deep learning, the rectified linear unit (or "ReLU") has been shown to be relatively efficient and is deemed to be computationally efficient. In essence, this function would use the representation, $y = \max(0, x)$, which ensures that we only need to make use of positive values for the weights, as it zeros out all negative values. Such a function is represented by figure 14.

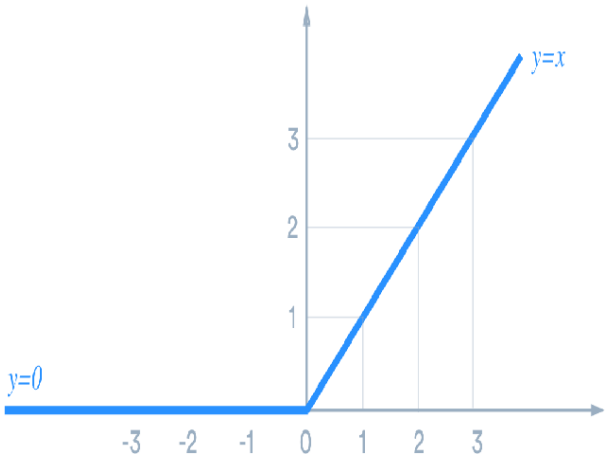

Figure 14: **Relu activation function**

The ReLU is the most commonly used activation function in neural networks so if you are unsure what activation function to use this is a good first choice, since:

- cheap to compute as there is no complicated math - model can therefore take less time to train or run
- converges faster - linearity means that the slope doesn't plateau, or "saturate" when x gets large
- doesn't have the vanishing gradient problem suffered by other activation functions like sigmoid or tanh
- sparsely activated - since it is zero for all negative inputs

However, one should be aware that there are several alternatives to the traditional ReLU, which include the Leaky ReLU, the Parametric ReLU, the Exponential Linear (ELU, SELU) and Concatenated ReLU. These are represented in figures 15, 16, and 17.

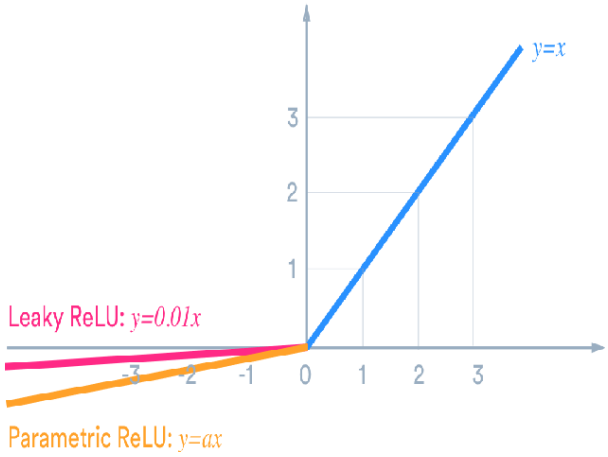

Figure 15: **Leaky ReLU & Parametric ReLU (PReLU)**

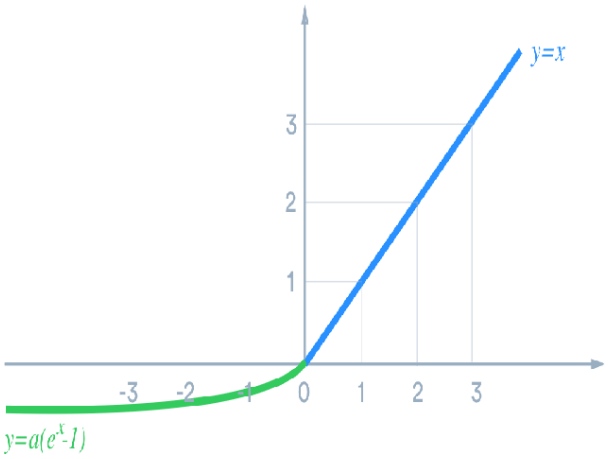

Figure 16: **Exponential Linear (ELU, SELU)**

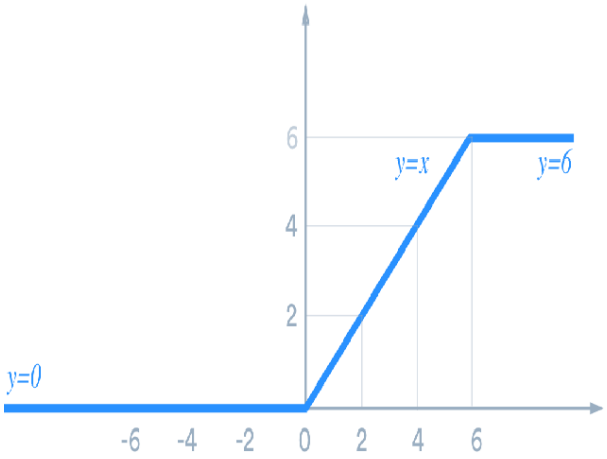

Figure 17: **Concatenated ReLU (CReLU)**

Note that when moving from the last dense layer to the output the activation layer would need to make use of specific functional forms to ensure that the output is appropriate. To return a probability one would need to use a sigmoid or softmax function (which takes the form of a logistic function) or in the case of a regression problem, we would simply exclude the activation function in the last dense layer, which would ensure that it is not subject to any transformation and would take the functional form of the input data.

4 Deep learning for time series

While much of the literature on applied deep learning methods focuses on image recognition there are a number of successful applications that have made use of time series data. Most of these applications make use of recurrent neural networks that process sequences of inputs one at a time, where each layer in the stack should return the full sequence of its outputs prior to the last layer. Of the different RNN methods, the Long Short-Term Memory (LSTM) algorithm usually provides improved results for relatively long sequences and is a fundamental tool for problems that involve time series processes. Unfortunately, although this algorithm is the one of the most powerful RNNs it is also the more expensive (in terms of the computational resources that are required). For this reason, some researchers make use of Gated recurrent unit (GRU) algorithm, which utilises similar principles to the LSTM, but is less sophisticated and as a result, less expensive in terms of the required computations.

The LSTM algorithm changes the way that past information is kept, where each unit keeps long-term information (protecting it from recursive operations). It then decides whether the current information matters or not before it clears whatever may not be useful anymore. In addition, this algorithm will also usually allow the RNN to produce output whenever it is ready to do so (i.e. when it satisfies an improvement in the loss criteria).

4.1 Data representation in tensor flow

Time series data is usually organised in a two-dimensional structure, where observations for each period of time are usually captured in the rows of a table or spreadheet, while the columns contain data the different variables that could be included in the model. When working with a deep learning model, this data must be transformed to a three-dimensional structure, or tensor.

Since the model needs to learn from the data, through a number of repeated unique opportunities, we need to structure the data appropriately. For example, in a simple hypothetical setting where we have ten observations for a univariate time series, it may be the case that we only need 3 autoregressive observations to produce a one-step ahead forecast. This would allow us to create a total of 8 different forecasting experiments, where we make use of observations y_{t-8}, y_{t-9} and y_{t-10} to learn something about observation y_{t-7}. Thereafter, we could make use of observations y_{t-7}, y_{t-8} and y_{t-9} to learn something about y_{t-6}, and so on.

$$
\begin{aligned}
y_{t-7} &\leftarrow y_{t-8}, \quad y_{t-9}, \quad y_{t-10} \\
y_{t-6} &\leftarrow y_{t-7}, \quad y_{t-8}, \quad y_{t-9} \\
y_{t-5} &\leftarrow y_{t-6}, \quad y_{t-7}, \quad y_{t-8} \\
y_{t-4} &\leftarrow y_{t-5}, \quad y_{t-6}, \quad y_{t-7} \\
y_{t-3} &\leftarrow y_{t-4}, \quad y_{t-5}, \quad y_{t-6} \\
y_{t-2} &\leftarrow y_{t-3}, \quad y_{t-4}, \quad y_{t-5} \\
y_{t-1} &\leftarrow y_{t-2}, \quad y_{t-3}, \quad y_{t-4} \\
y_{t} &\leftarrow y_{t-1}, \quad y_{t-2}, \quad y_{t-3}
\end{aligned}
$$

Each one of these forecasting experiments constitutes a sample or batch, while the number of observations that are used within each sample may be termed a time step. In this case, each sample contains 3 time steps. In the machine learning literature the regressors are termed features. Hence, in a univariate case we only have one feature, while in a multivariate setting we could include more than one feature. Of course it would not be necessary to include all of the above batches, and we could select a few of these batches, which would reduce computational time.

The procedure for converting a time series or group of time series variables into a three-dimensional tensor is displayed in figure 18.

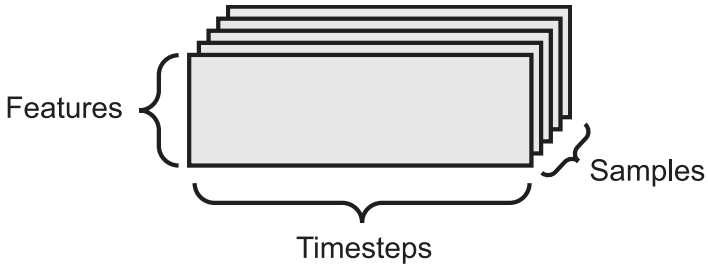

Figure 18: **Time series data in tensors**

5 Useful tools and packages

Most of the deep learning tools have been developed in Python and C++. The most popular of these tools is currently *Tensorflow*, which has been developed by Google. This code has been made available to the public and Google also provide an API that goes by the name of *Keras*, which simplifies the coding task. There are **R** packages for both *Tensorflow* and *Keras* and there are already a number of examples that you can learn from. You could also make use of one of the many alternative packages in Python - such as Pytorch, Torch, Theano, Caffe, etc. However, before you commit a huge amount of resources to learning about these packages, you may want to consider the results in figure 19, which displays the number of Google web search for different deep-learning frameworks.

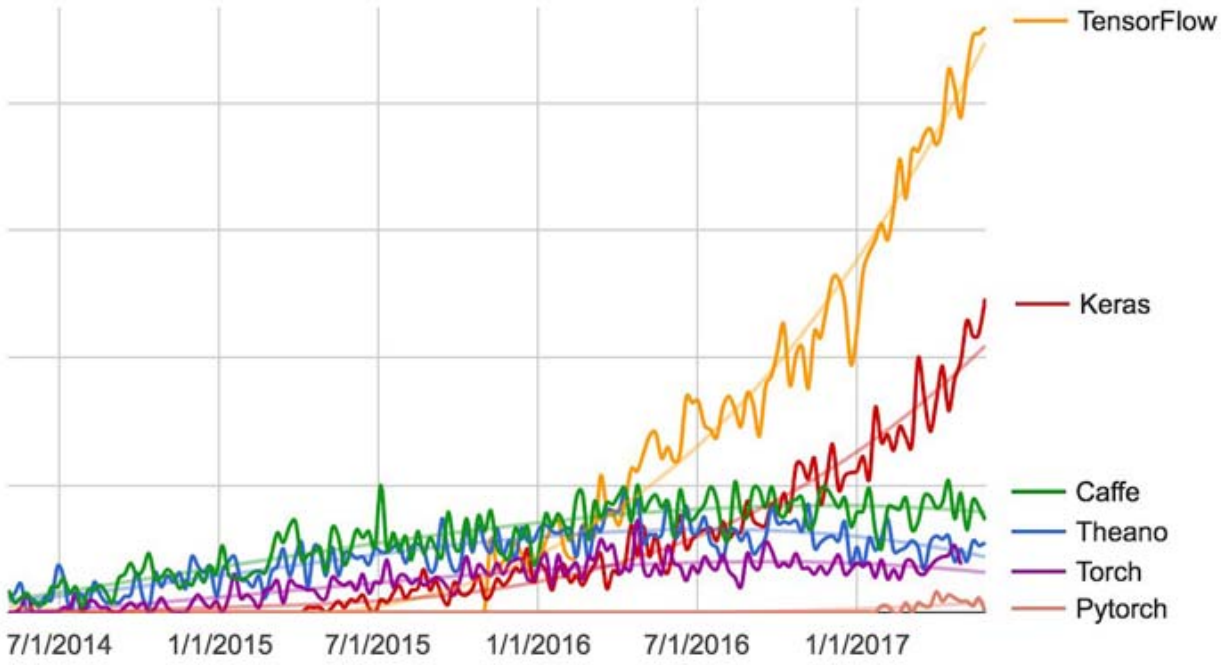

Figure 19: **Google web search for different deep-learning frameworks**

6 Conclusion

There are many potentially interesting applications for deep learning methods within the field of time series analysis. And at the current point in time, there is huge amount of testing on different applications still needs to be performed. Note that the application of these techniques generally requires large amounts of data and this has also changed the way in which we now collect data. As we continue to apply these techniques to different problems, it will require a huge amount of computing power. Note also that it is unlikely that this technique will be the preferred method for every potential machine learning application. However, it has already shown that it is capable of producing impressive results in a number of applications.

7 References

1. A. M. Turing, “Computing Machinery and Intelligence”, Mind 59, no. 236 (1950): 433-460.↩