# AI for business: a glossary

Natalia Efremova

*University of Oxford, Said Business School*

**Abstract**

This document gives a high-level overview of the main AI terms and serves as starting point for further research. We provide more technical details on the topics, which are relevant to the course, while skipping very basic ones and more difficult subjects. All the references are available in the text via hyperlinks.

**Contents**


**1 AI: definitions** **4**
1.1 Overview of AI field . . . . . 4
1.1.1 Rule-based systems . . . . . 4
1.1.2 Machine learning . . . . . 5
1.1.3 Deep learning . . . . . 7
1.2 "L" for learning . . . . . 7
1.2.1 Supervised learning . . . . . 8
1.2.2 Training and testing . . . . . 9
1.2.3 Unsupervised learning . . . . . 11
1.2.4 Reinforcement learning . . . . . 11
1.3 Choice of ML models . . . . . 12

**2 ML applications** **14**
2.1 Computer vision . . . . . 14
2.1.1 Neural Networks: a formal definition . . . . . 15
2.1.2 Neural network training . . . . . 17
2.1.3 Neural network regularisation . . . . . 20
2.1.4 Convolutional Neural Networks . . . . . 20
2.2 Natural language processing . . . . . 21
2.2.1 Recurrent Neural Networks . . . . . 25
2.2.2 Transformer models . . . . . 28
2.3 Generative adversarial networks . . . . . 29

**3 AI deployment** **31**
3.1 Data component . . . . . 31
3.2 ML for feature engineering . . . . . 31
3.2.1 Pearson correlation . . . . . 32
3.2.2 Principal components analysis . . . . . 33
3.3 Development component . . . . . 34
3.4 Machine learning frameworks . . . . . 35
3.4.1 Scikit-learn . . . . . 35
3.4.2 TensorFlow . . . . . 36
3.4.3 Keras . . . . . 37
3.4.4 XGBoost . . . . . 37
3.4.5 PyTorch . . . . . 37

**4 Risks, biases and adversarial attacks** **38**

**5 Build versus buy** **38**

**Glossary** **40**

## 1. AI: definitions

The aim of this book is to create a high-level description of Machine learning (ML) methods for students taking a course on artificial intelligence (AI) and/or analytics in Business Schools. It contains description of main topics in AI, from academic research to practical deployment in business. This book consists of three main parts. First part focused on categories of artificial intelligence approaches and how they are related, types of machine learning and the basics of ML training and testing. Second part outlines main applications in business (such as computer vision and natural language processing) and the specifics of implementation (such as feature engineering and machine learning frameworks). This short introduction is followed by an extended glossary of most important terms in AI. Also it includes links to additional sources in the text of the document itself.

### *1.1. Overview of AI field*

This section is based on the book by Ian Goodfellow, Yoshua Bengio and Aaron Courville "Deep Learning": it is available online.

A fundamental goal of research in **Artificial intelligence** (AI) is building machines that can perceive, analyse and interpret the world. But the world is very complex, with multiple sources of information that need to be combined, such as visual, audio, tactile and temporal data. Therefore, within AI, there are multiple sub-fields that work with different kinds of problems and inputs. For example, computer vision focuses on the theory and technology that enables machines to see by processing and extracting information from imaging data, such as photos and videos. Natural language processing extracts information from text data. Audio processing addresses the problem of transferring audible speech into text and vice versa. Within these sub-fields of AI, there are also different types of algorithms, with different structure and purpose. A diagram on Fig.1 shows categories of artificial intelligence approaches and how they are related.

The term "artificial intelligence" combines multiple approaches, including knowledge-based and rule-based algorithms and machine learning (ML).

#### *1.1.1. Rule-based systems*

**Rule-based systems** make inference from the data according to the predefined rules (called "propositions" or simply "if-then" rules). For example, traditional medical diagnostic systems are rule-based. Rules in rule-bases

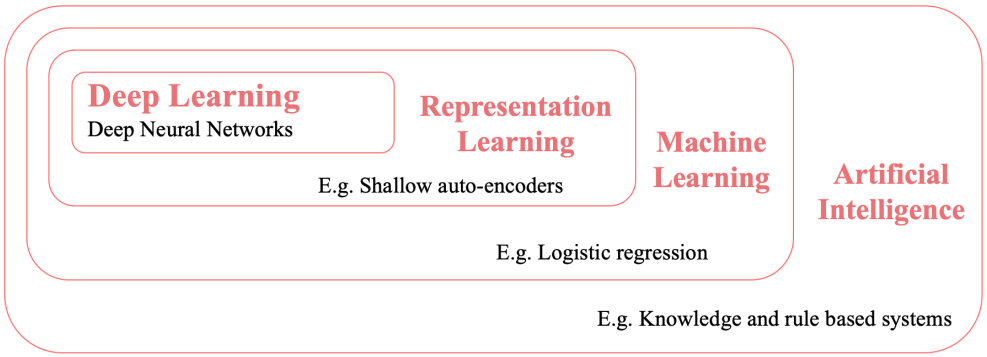


Figure 1: Categories of artificial intelligence approaches and how they are related.

systems are created manually, from expert opinions, or semi-automatically, from existing data combined with expert opinions. Although this part of AI is the oldest one, only a small proportion of commercial systems works without some type of rules at least in one component of the system.

*1.1.2. Machine learning*

**Machine learning algorithm** learn from data automatically, given a substantial number of samples from a particular sub-field of AI as inputs. This is a large class of AI problems, with the the learning process being the only similarity between them. The most common algorithm in ML is regression models, for example logistic regression. One of the largest ML paradigms in ML is supervised learning, when we input data as well as the answers expected from the data, and output the rules. These rules can then be applied to new data to produce original answers (Fig.2).

Regression models posses the ability to learn features automatically from data and, at the same time, they are easily **interpretable**: i.e. they allow **direct interpretation** of the trained parameters. Interpretability of a ML model is of extreme importance, since the features that they learn can be easily assessed and the results, obtained with these features, can be explained. The requirement of the model outputs to be explained often conditions the choice which model to choose.

**Probabilistic modelling** is the application of the principles of statistics to data analysis. It was one of the earliest forms of machine learning, and it's still widely used to this day. One of the best-known algorithms in this category is the Naive Bayes algorithm. **Naive Bayes**is a type of machine-

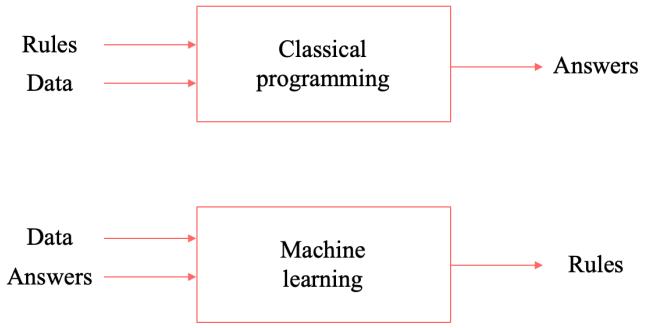


Figure 2: Machine learning: a new programming paradigm. In supervised learning, we input data as well as the answers expected from the data, and output the rules. These rules can then be applied to new data to produce original answers (from F.Chollet 2018).

learning classifier based on applying Bayes' theorem while assuming that the features in the input data are all independent (a strong, or "naive" assumption, which is where the name comes from). This form of data analysis predates computers and was applied by hand decades before its first computer implementation (most likely dating back to the 1950s). Bayes' theorem and the foundations of statistics date back to the eighteenth century, and these are all you need to start using Naive Bayes classifiers.

The performance of a simple ML model, such as regression or Naive Bayes, depends heavily on the **Representation** of the data they are given (Fig.3). For example, when logistic regression is used to recommend cesarean delivery, the AI system does not examine the patient directly. Instead, the doctor tells the system several pieces of relevant information, such as the presence or absence of a uterine scar. Each piece of information included in the representation of the patient is known as a **Feature**. Logistic regression learns how each of these features of the patient correlates with various outcomes (from Goodfellow et al., 2016). Many AI tasks can be solved by designing the right set of features to extract for that task, then providing these features to a simple ML algorithm. This process is called **feature engineering**.

For many tasks, however, it is difficult to know what features should be extracted. One solution to this problem is to use machine learning to discover not only the mapping from representation to output but also the representation itself. This approach is known as **Representation learning** (Fig.4). Learned representations often result in much better performance than can be obtained with hand-designed representations. They also enable

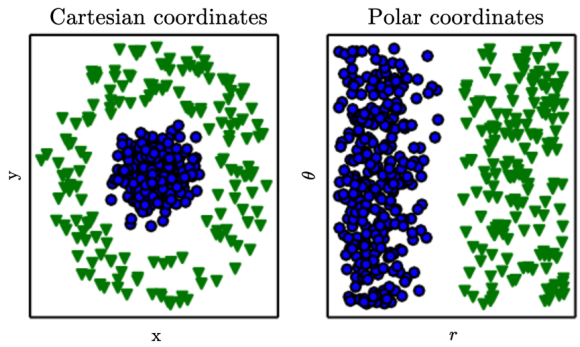


Figure 3: Example of different representations: suppose we want to separate two categories of data by drawing a line between them in a scatterplot. In the plot on the left, we represent some data using Cartesian coordinates, and the task is impossible. In the plot on the right, we represent the data with polar coordinates and the task becomes simple to solve with a vertical line (from Goodfellow et al., 2016).

AI systems to rapidly adapt to new tasks, with minimal human intervention.

*1.1.3. Deep learning*

**Deep learning** is a sub-field of machine learning that works with more sophisticated algorithms that require very large amounts of data. Deep learning models, also known as neural networks, are the most complex algorithms in the field of AI. They are often the most difficult to train, the least interpretable and require the vast amounts of data. Due to the lack of transparency and difficulty to explain the result, they are often referred to as "black box" approach. However, they allow working with the most complex data and achieve the best performance on noisy data.

*1.2. "L" for learning*

In this section, I will describe types of machine learning and give a few useful definitions. Most of the definitions are taken from Andrew Ng's course on Machine Learning from Stanford University. All the lectures are available on YouTube and we encourage you to watch them if you want to learn more on ML.

A section on reinforcement learning is based on a book by Sutton and Barto, which is also available online.

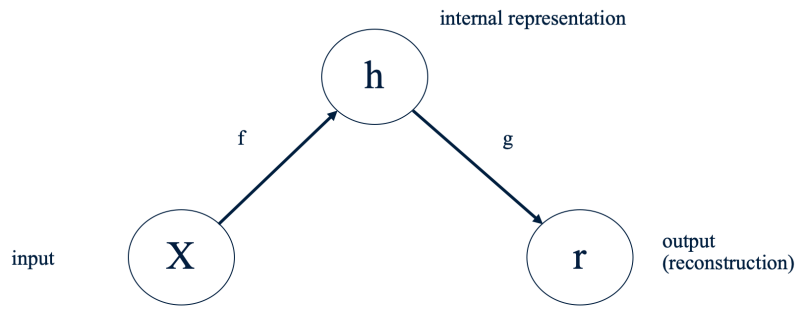


Figure 4: Example of representation learning: autoencoder model. The general structure of an autoencoder, mapping an input x to an output (called reconstruction) r through an internal representation or code h. The autoencoder has two components: the encoder f (mapping x to h ) and the decoder g (mapping h to r).

*1.2.1. Supervised learning*

**Supervised Learning**. In supervised learning, we are given a data set and already know what our correct output should look like, having the idea that there is a relationship between the input and the output.

To describe the supervised learning problem slightly more formally, our goal is, given a training set, to learn a function $h : X \to Y$ so that $h(x)$ is a good predictor for the corresponding value of $y$. Function $h$ is called a hypothesis (Fig.5).

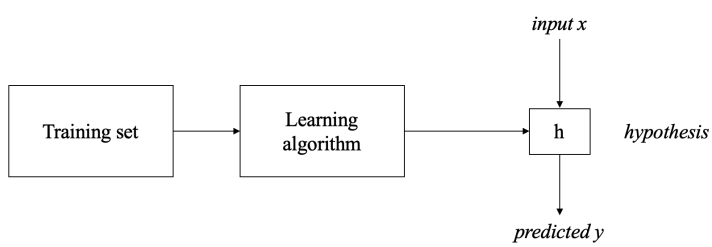


Figure 5: Supervised learning.

In the majority of ML texts, you will see the following notation: $x(i)$

denotes the input variables, also called input features, and $y(i)$ denotes the output or target variable that we are trying to predict. A pair $(x(i); y(i))$ is called a training example, and the dataset that we'll be using to learn a list of n training examples is called a training set.

During training of a ML algorithm, we adjust parameters $\theta$ in a way that our model will get better predictions each time we calculate an output of the model (Fig.6).

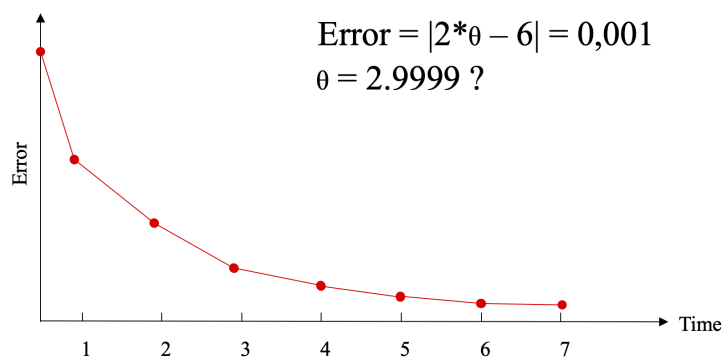


Figure 6: Training process: the goal of the training process is to minimise the error.

When the target variable that we're trying to predict is continuous, (e.g. house price) we call the learning problem a regression problem. When y can take on only a small number of discrete values (such as 'cats' or 'not cats'), we call it a classification problem.

*1.2.2. Training and testing*

**Training and test sets.** To evaluate the performance of our model, we need to run it on the previously unseen data. For this reason, we usually divide the whole dataset into two subsets: training set and test set. We train our model on the training set and we evaluate it's performance on the test set. For this purpose, we calculate the model on each data point in the test set and estimate average performance. In the simple case, we calculate the difference between the real output and the desired output (the label for this data point), which is called an error. We say that the average error that we obtain on the test set is the performance of the model. Test set is usually

much smaller that training set (e.g. 80% of the data is training set, and 20% of the data is test set). However, this is not always the case. When our dataset is very large (millions of data points), the test set would be smaller.

Another useful notion in the training and testing framework is **Validation set** (from the best book in pattern recognition, which is available via this link). In the case we can build many models (we have enough data) we may wish to consider a range of different types of model in order to find the best one for our particular application. One approach is simply to use some of the available data to train a range of models, or a given model with a range of values for its complexity parameters, and then to compare them on independent data, sometimes called a validation set, and select the one having the best predictive performance. If the model design is iterated many times using a limited size data set, then some over-fitting to the validation data can occur and so it may be necessary to keep aside a third test set on which the performance of the selected model is finally evaluated.

In many applications, however, the supply of data for training and testing will be limited, and in order to build good models, we wish to use as much of the available data as possible for training. However, if the validation set is small, it will give a relatively noisy estimate of predictive performance. One solution to this dilemma is to use cross-validation, which is illustrated in Fig.7. This allows a proportion $(S-1)/S$ of the available data to be used for training while making use of all of the data to assess performance. When data is particularly scarce, it may be appropriate to consider the case $S = N$, where $N$ is the total number of data points, which gives the leave-one-out technique.

**Overfitting** is a very important concept in ML. Whet we say that the model overfits, we mean that it's fitting the training data too well, i.e. the noise or random fluctuations in the training data is picked up and learned as concepts by the model. The problem is that these concepts do not apply to new data and negatively impact the models ability to generalise. To avoid this problem, regularisation is widely applied in ML models. **Regularization** is a technique used for tuning the function by adding an additional penalty term in the error function. The additional term controls the excessively fluctuating function such that the coefficients don't take extreme values.

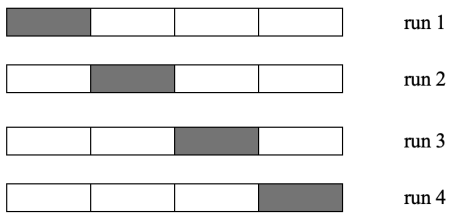


Figure 7: The technique of S-fold cross-validation, illustrated here for the case of $S = 4$, involves taking the available data and partitioning it into S groups (in the simplest case these are of equal size). Then $S-1$ of the groups are used to train a set of models that are then evaluated on the remaining group. This procedure is then repeated for all $S$ possible choices for the held-out group, indicated here by the red blocks, and the performance scores from the $S$ runs are then averaged.

### *1.2.3. Unsupervised learning*

**Unsupervised Learning** allows us to approach problems with little or no idea what our results should look like. We can derive structure from data where we don't necessarily know the effect of the variables. We can derive this structure by clustering the data based on relationships among the variables in the data, such as proximity of data points. With unsupervised learning there is no feedback based on the prediction results.

This type of machine learning is called "unsupervised" because we're not telling the algorithm in advance that these are type 1 data, type 2 data or type 3 data like in supervised learning. Instead we are showing a bunch of data and ask to automatically find structure in the data (Fig.8).

### *1.2.4. Reinforcement learning*

**Reinforcement Learning** is focused on goal-directed learning from interaction between a learning agent and its environment (Sutton and Barto: reinforcement learning). Such an agent must be able to sense the state of the environment to some extent and must be able to take actions that affect the state. The agent also must have a goal or goals relating to the state of the environment.

One of the challenges that arise in reinforcement learning is the trade-off between exploration and exploitation. To obtain a lot of reward, a reinforcement learning agent must prefer actions that it has tried in the past and found to be effective in producing reward. But to discover such actions, it has to try actions that it has not selected before. The agent has to exploit

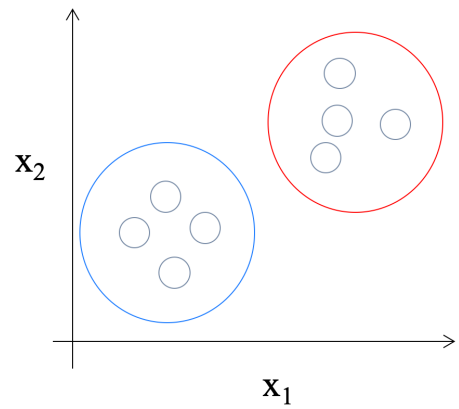


Figure 8: Unsupervised learning: red and blue circles represent clustering ofidentical datapoints by proximity.

what it already knows in order to obtain reward, but it also has to explore in order to make better action selections in the future. The dilemma is that neither exploration nor exploitation can be pursued exclusively without failing at the task. The agent must try a variety of actions and progressively favour those that appear to be best.

There are four main elements of a reinforcement learning system: a policy, a reward signal, a value function, and, optionally, a model of the environment. A policy defines the learning agent's way of behaving at a given time. A reward signal defines the goal in a reinforcement learning problem. On each time step, the environment sends to the reinforcement learning agent a single number, a reward. The agent's sole objective is to maximise the total reward it receives over the long run. A value function specifies what is good in the long run. A model of the environment is something that mimics the behaviour of the environment, or more generally, that allows inferences to be made about how the environment will behave (Fig.9).

*1.3. Choice of ML models*

Within the field of machine learning, there are no "good" or "bad" models. The choice of what model to use in which case depends on multiple parameters. I will briefly outline the main questions, that one might consider to make the right choice of the model.

- Amount of data. This is by far the most important consideration for the data scientist in the choice of a model. Some algorithms require

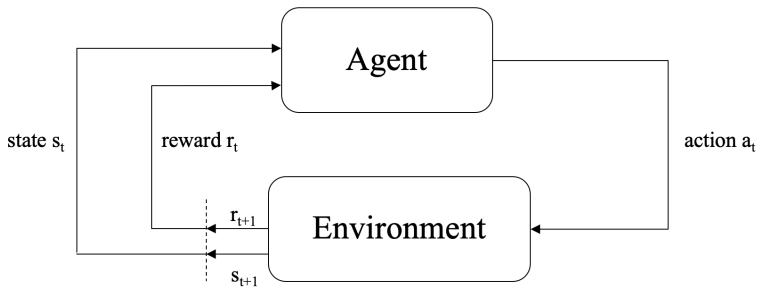


Figure 9: Reinforcement learning.

just a few data points (e.g. regression models can work with around 100 data entries, whereas neural network model will require thousands of examples)

- Data structure. There are models that are known to work better for computer vision problems or image data, numeric data, text data or time series data. Some approaches allows working with unstructured data or missing data, whereas others will require perfectly structured and normalised data. The structure of the output is as important as the structure of the input data. There are models that predict a single value within a range of values (regression models), a category (classification models), a distribution (probabilistic models) and so on.

- Model performance. The goal of machine learning model during training is to minimise the error after the training. ML algorithms differ in the ability to achieve best performance depending on the amount of data, complexity of the model etc. For example, in some settings regression models can achieve better performance than neural network models.

- Interpretability of the data and the cost of error. This is the second most important question that data scientists address while choosing the model: if the model makes a mistake, what will be the cost of it? Therefore in the fields like finance and medicine developers tend to use simpler models, which are more transparent and easier to explain.

- Scalability of a model. This constraint is related to a model's ability

to scale quickly from 100 data points to a million without a significant increase in computational complexity.

- Privacy, security and other considerations. There is a large number of less significant considerations, that might arise in each particular case.

## 2. ML applications

### *2.1. Computer vision*

**Computer vision** is an interdisciplinary scientific field that deals with how computers can gain high-level understanding from digital images or videos. From the engineering perspective, it seeks to understand and automate tasks that the human visual system can do.

From a ML perspective, computer vision is type of algorithms that works with image and video data to understand it's content. Sometimes, we also imply processing of information from other sensors, such as lidars, X-rays or satellite radar imagery.

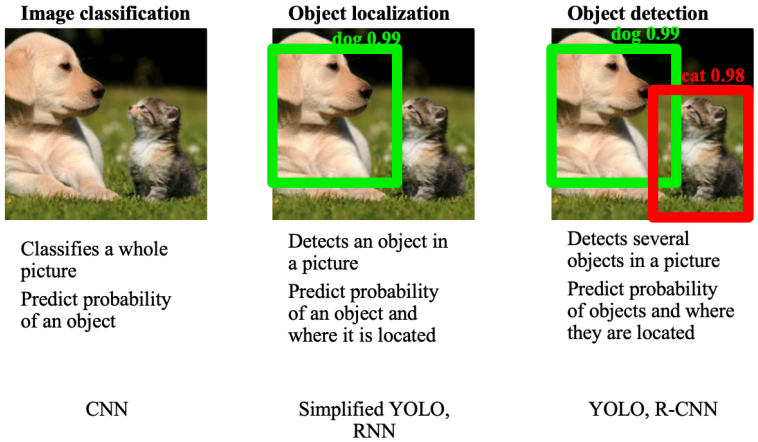


Figure 10: Types of computer vision applications.

For simplicity, we will consider working with images. First, we define the types of computer vision problems. If we want to classify a whole image, this is called an image classification problem. If we want to find an object on the

image, this is an object localisation problem. If we are searching for multiple objects, this is an object detection problem (Fig. 10).

Computer vision is one of the most complex problems in ML. The most commonly used type of an algorithm for this problem is neural network (Fig.11). The term 'neural network' has its origins in attempts to find mathematical representations of information processing in biological systems (from McCulloch and Pitts, 1943. Currently, neural networks have little to do with organisation of our visual system, but at the very beginning they took inspiration from biological neurons in primate brain.

*2.1.1. Neural Networks: a formal definition*

**Neural networks** are efficient models for statistical pattern recognition. They consist of multiple simple functions, that process input features in parallel to produce some output value.

Basic neural network model can be described a series of functional transformations. First, we construct $M$ linear combinations of the input variables $x_1, ..., x_n$ in the form

$$a_j = \sum_{i=1}^{D} \omega_{ji}^{(k)} x_i + \omega_{j0}^{(k)} \tag{1}$$

where $j = 1, ..., M$, and the superscript $(k)$ indicates that the corresponding parameters are in the $k$-th 'layer' of the network. We refer to the parameters $\omega_j^{(1)} i$ as weights and the parameters $\omega_{j0}^{(1)}$ as biases. The quantities $a_j$ are known as activations. Each of them is then transformed using a differentiable, nonlinear activation function $h$ to give

$$z = h(a_j) \tag{2}$$

These quantities correspond to the outputs of the functions that, in the context of neural networks, are called hidden units.

The nonlinear functions $h(a_j)$ are generally chosen to be sigmoidal functions such as the logistic sigmoid or the hyperbolic tangent function.These values are again linearly combined to give output unit activations $a_k$. For regression problems, the activation function is the identity so that $y_k = a_k$. Similarly, for multiple binary classification problems, each output unit activation is transformed using a logistic sigmoid function so that

$$\hat{y}_k = \sigma(a_k) \tag{3}$$

where

$$\sigma(a_k) = \frac{1}{1 + exp^{-a}} \tag{4}$$

Finally, for multi-class classification problems, a softmax activation function is used:

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{5}$$

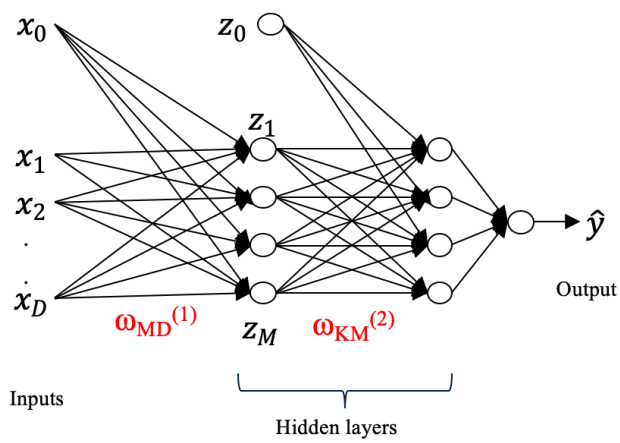


Figure 11: Basic neural network model with two hidden layers. $\omega$ indicates trainable parameters at each layer.

In the neural network, described above, each arrow represent a **Parameter**, which is modified slightly during the training process. The output of a neural network is calculated during the forward propagation process, whereas the weights are adjusted during the back propagation process (Fig.12). In back propagation, we calculate the change of the weights (or parameters) of the model based on the error $Err$ = desired output $y$ - output of the network $\hat{y}$, that was calculated in the forward propagation process.

The number of hidden layers describes the depth of a neural network. If there is only one hidden layer, the model is called shallow neural network. If the model has two or more layers, it is called deep neural network. The sub-field of machine learning that studies deep neural networks is called deep learning.

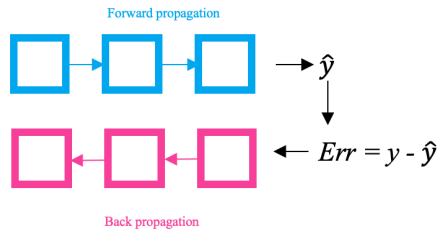


Figure 12: Forward and back propagation in neural network.

*2.1.2. Neural network training*

Training a neural network involves optimisation of parameters $\omega$ at each layer of neural network. This involves finding a weight vector $\omega$ which minimises the chosen function $E(\omega)$. However, the error function typically has a highly nonlinear dependence on the weights and bias parameters. Therefore, we take a gradient of this function and in small steps move towards the direction where the gradient vanishes. It is possible to evaluate the gradient of an error function efficiently by means of the back propagation procedure.

The simplest approach to using gradient information is to choose the weight update to comprise a small step in the direction of the negative gradient, so that

$$\omega^{(\tau+1)} = \omega^{(\tau)} - \eta \nabla E(\omega^{(\tau)}) \tag{6}$$

where $\tau$ is a time step and $\eta > 0$ is a learning rate.

After each such update, the gradient is re-evaluated for the new weight vector and the process repeated. The error function is defined with respect to a training set, and so each step requires that the entire training set be processed in order to evaluate $\nabla E$. This training algorithm is known as batch gradient descent. On-line gradient descent, also known as sequential gradient descent or stochastic gradient descent, makes an update to the weight vector based on one data point at a time.

To evaluate the gradient of the error function, we use a local message passing scheme in which information is sent alternately forwards and backwards through the network and is known as error backpropagation, or sometimes simply as **Backpropagation**. Most ML training algorithms involve an iterative procedure for minimisation of an error function, with adjustments to the weights being made in a sequence of steps. At each such step, we can distinguish between two distinct stages. In the first stage, the derivatives of the error function with respect to the weights must be evaluated. The

important contribution of the back propagation technique is in providing a computationally efficient method for evaluating such derivatives.

We now derive the backpropagation algorithm for a general neural network, with any differentiable nonlinear activation functions and a broad class of error functions.

In a general feed-forward network, each unit computes a weighted sum of its inputs of the form

$$a_j = \sum_i \omega_{ji} z_i \tag{7}$$

where $z_I$ is the activation of a unit, or input, that sends a connection to unit $j$, and $\omega_{ji}$ is the weight associated with that connection (for simplicity, here we consider biases to be a part of weight matrix and do not show them explicitly). Then this sum is transformed by a nonlinear activation function $h$ to give the activation $z_j$ of unit $j$ in the form

$$z_j = h(a_j) \tag{8}$$

For each pattern in the training set, we shall suppose that we have supplied the corresponding input vector to the network and calculated the activations of all of the hidden and output units in the network by successive application of two equations above. This process is often called forward propagation because it can be regarded as a forward flow of information through the network.

To calculate error $E$, we evaluate of the derivative of $E_n$ with respect to a weight $\omega_{ji}$. $E_n$ depends on the weight $\omega_{ji}$ only via the summed input $a_j$ to unit $j$. We can therefore apply the chain rule for partial derivatives to give

$$\frac{\partial E_n}{\partial \omega_{ji}} = \frac{\partial E_n}{\partial a_{ji}} \frac{\partial a_j}{\partial \omega_{ji}}. \tag{9}$$

We simplify this notation by introducing

$$\delta_j = \frac{\partial E_n}{\partial a_{ji}}. \tag{10}$$

Using formula 7, we can write

$$\frac{\partial E_n}{\partial a_j} = z_i, \tag{11}$$

therefore we obtain

$$\frac{\partial E_n}{\partial \omega_{ji}} = \delta_j z_i. \tag{12}$$

The backpropagation formula can be written as following:

$$\delta_j = h'(a_j) \sum_k \omega_{kj} \delta_k, \tag{13}$$

which tells us that the value of $\delta$ for a particular hidden unit can be obtained by propagating the $\delta$'s backwards from units higher up in the network, as illustrated in Fig. 13.

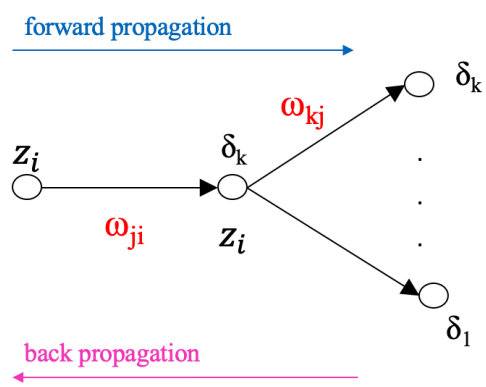


Figure 13: Illustration of the calculation of $\delta_j$ for hidden unit $j$ by backpropagation of the $\delta$'s from those units $k$ to which unit $j$ sends connections. The blue arrow denotes the direction of information flow during forward propagation, and the pink arrow indicates the backward propagation of error information.

The backpropagation procedure can therefore be summarised as follows.

1. Apply an input vector $x_n$ to the network and forward propagate through the network to find the activations of all the hidden and output units.
2. Evaluate the $\delta_k$ for all the output units
3. Backpropagate the $\delta$'s to obtain $\delta_j$ for each hidden unit in the network.
4. Evaluate the required derivatives.

For batch methods, the derivative of the total error $E$ can then be obtained by repeating the above steps for each pattern in the training set and then summing over all patterns.

*2.1.3. Neural network regularisation*

Similar to other ML models, we need to control the complexity of a neural network model in order to avoid overfitting. One way to control complexity by the addition of a regularisation term to the error function.

There are multiple alternative approaches to neural nerwork regularisation, e.g. dropout regularisation. Dropout is a regularisation method that approximates training a large number of neural networks with different architectures in parallel. During training, some number of layer outputs are randomly ignored or "dropped out." This has the effect of making the layer look-like and be treated-like a layer with a different number of nodes and connectivity to the prior layer. In effect, each update to a layer during training is performed with a different "view" of the configured layer.

By dropping a unit out, we temporarily remove it from the feed-forward and feed-back calculations, along with all its incoming and outgoing connections. Dropout has the effect of making the training process noisy, forcing nodes within a layer to probabilistically take on more or less responsibility for the inputs.

This conceptualisation suggests that dropout breaks-up situations where network layers co-adapt to correct mistakes from prior layers, in turn making the model more robust.

*2.1.4. Convolutional Neural Networks*

Computer vision is one of the areas that's been advancing rapidly thanks to deep learning. Deep learning computer vision is now helping self-driving cars figure out where the other cars and pedestrians around so as to avoid them.

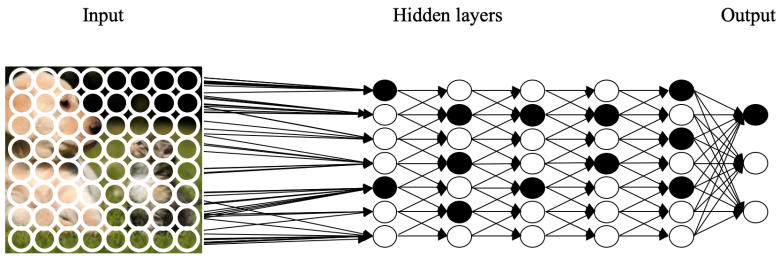


Figure 14: Neural network for image recognition. Each pixel (or 3 by 3 pixel square) serves as an input of our neural network.

One of the challenges of computer vision problems is that the inputs can get really big. If we consider a 1000 pixel by 1000 pixel image, the dimension of the input features will be 1000 by 1000 by 3, because you have three RGB channels (Fig.15), which results in three million input features. That will make the weight matrix of the neural network very big, with billions of parameters. To avoid that, we need to implement the convolution operation, which is one of the fundamental building blocks of convolutional neural networks.

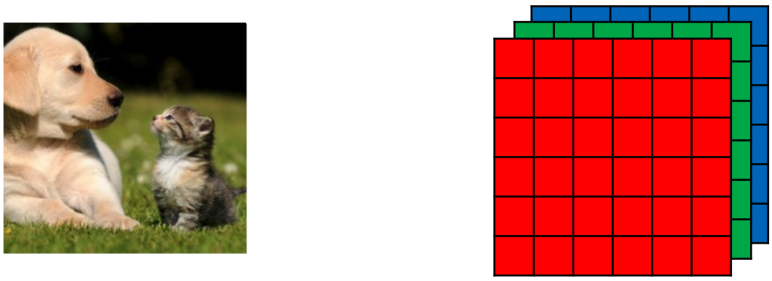

Figure 15: Representation of an image as RGB matrix: each colour image consists of three matrices: red, green and blue colours.

The convolution operation is one of the fundamental building blocks of a convolutional neural network. The 2D convolution is a fairly simple operation: you start with a kernel (or a filter), which is a small matrix of weights. This kernel "slides" over the 2D input data, performing an element-wise multiplication with the part of the input it is currently on, and then summing up the results into a single output pixel (Fig.16). The kernel repeats this process for every location it slides over, converting a 2D matrix of features into yet another 2D matrix of features. The numbers in the kernel on the first pass are defined randomly, but then trained in the learning process with backpropagation algorithm in a way each filter 'learns' to detect some feature, which is useful for our recognition process.

A convolutional neural network consists of a few convolutional layers and usually one or two fully connected layers, which are similar to those that we have discussed in previous section (Fig.17).

*2.2. Natural language processing*

The best book to refer to on NLP can be found via this link. The easier version is this one: Deep Learning with Python, which has a chapter on

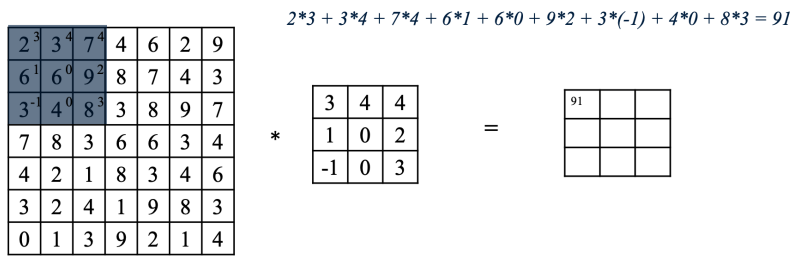


Figure 16: Convolution operation.

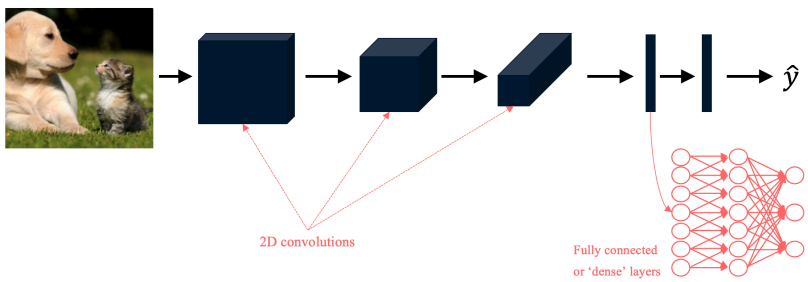


Figure 17: Convolutional neural network.

natural language processing.

**Natural language processing** (NLP) is a subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, in particular how to program computers to process and analyse large amounts of natural language data.

**Deep learning for natural-language processing** is pattern recognition applied to words, sentences, and paragraphs, similarly to how computer vision is pattern recognition applied to pixels. Like all other neural networks, deep-learning models don't take as input raw text: they only work with numeric data. Vectorizing text is the process of transforming text into numeric Tensor. This can be done in multiple ways:

- Segment text into words, and transform each word into a vector.
- Segment text into characters, and transform each character into a vector.
- Extract n-grams of words or characters, and transform each n-gram

into a vector. N-grams are overlapping groups of multiple consecutive words or characters.

The different units into which you can break down text (words, characters, or n-grams) are called tokens, and breaking text into such tokens is called tokenization. All text vectorization processes consist of applying some tokenization scheme and then associating numeric vectors with the generated tokens. These vectors, packed into sequence tensors, are fed into deep neural networks. There are multiple ways to associate a vector with a token. There two major ones: one-hot encoding of tokens, and token embedding (typically used exclusively for words, and called word embedding).

One-hot encoding consists of associating a unique integer index with every word and then turning this integer index $i$ into a binary vector of size $N$ (the size of the vocabulary); the vector is all zeros except for the $i$-th entry, which is 1 (Fig.18).

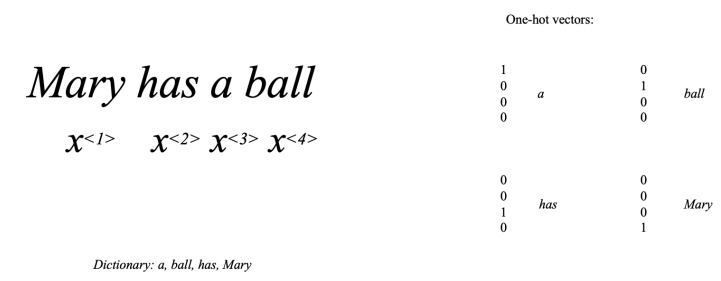


Figure 18: Text vectorization: one-hot encoding.

Another popular and powerful way to associate a vector with a word is the use of dense word vectors, also called **Word embeddings**. Whereas the vectors obtained through one-hot encoding are binary, sparse (mostly made of zeros), and very high-dimensional (same dimensionality as the number of words in the vocabulary), word embeddings are low-dimensional floating-point vectors (that is, dense vectors, as opposed to sparse vectors).

There are two ways to obtain word embeddings:

- Learn word embeddings jointly with the main task you care about (such as document classification or sentiment prediction). In this setup, you

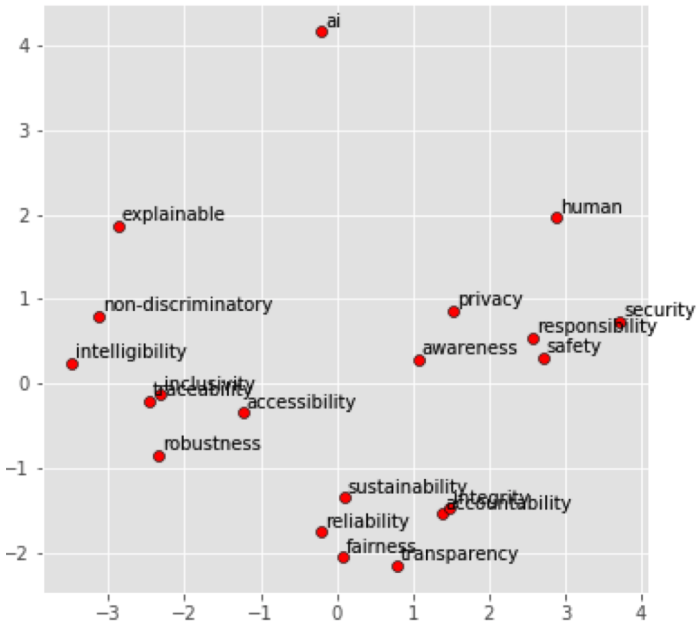


Figure 19: Text vectorization: visualising word embeddings (from one of the FOMI projects).

start with random word vectors and then learn word vectors in the same way you learn the weights of a neural network.

- Load into your model word embeddings that were pre-computed using a different machine-learning task than the one you're trying to solve. These are called pre-trained word embeddings.

There multiple mathematical approaches, that can be used for finding word vectors. The most popular one is Word2Vec approach. Technical details of this approach are outside of the scope of this document, but we encourage you to learn more if you are interested in NLP.

Trained word embeddings can be visualised on the 2D plane (Fig.18). After the text was turned into vectors, it can be processed with a neural network, similarly to images. However, in NLP not only a single word matters, but a sequence of words. Therefore, we employ a different type of models, which takes into account sequential information.

*2.2.1. Recurrent Neural Networks*

So far, out discussion about neural network was about elements that do not have temporal structure, such as images and numbers. However, when it comes to language processing, this is not going to work: the meaning of words that we say depend on what we have said before and, sometimes, on what we are going to say after. This dependence in neural networks is called a recurrent connection. If we consider the sentence *"Mary has a ball"*, we would start counting from the first word of the sentence, *Mary* (this will be our input $x^1$), and finish with the last, *ball* ($x^4$).

A Recurrent neural network (RNN) processes sequences by iterating through the sequence elements and maintaining a state containing information relative to what it has seen so far. In general case, an RNN will take as an input $n$ elements of a sequence and process it in $n$ steps. At each step, it will "memorise" it's inner state a and pass it to the next "pass" of a neural network. In a diagram, we often depict it as an "unfolded" sequence of $t$ steps, each of which will have and input $x^i$, output $\hat{y^i}$, incoming hidden state $a^{t-1}$ from a previous element and outgoing hidden state $a^t$ that will be used in the next element (Fig.20).

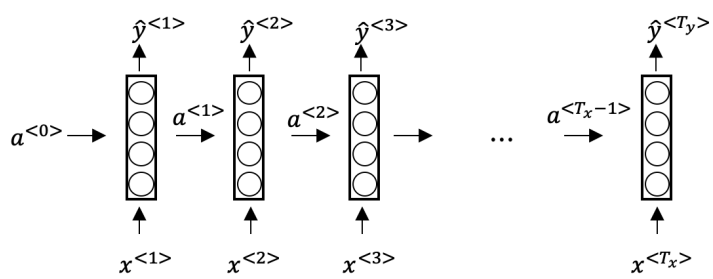


Figure 20: Simple recurrent neural network.

On the last time step our RNN inputs $x^{T_x}$ and then it outputs $\hat{y}^{T_y}$. In this example, a number of steps $T_x$ is equal to $T_y$, but the architecture will change a bit if $T_x$ and $T_y$ are not identical. So at each time step, the recurrent neural network that passes on as activation to the next time step for it to use. And to start the processing of the first unit, we'll also have some made-up activation at time zero, this is usually the vector of zeros. Some researchers will initialized $a^0$ randomly. You have other ways to initialize $a^0$ but really

having a vector of zeros as the fake times zero activation is the most common choice. So that gets input to the neural network.

The recurrent neural network scans through the data from left to right. The parameters it uses for each time step are shared. In order to train such neural network, we will also need a set of parameters. The parameters governing the connection from $x_1$ to the hidden layer, will be some set of parameters we're going to write as $\omega_{ax}$ and this is the same parameters that are used for every time step. The activations (the horizontal connections) will have some set of parameters $\omega_{aa}$ and the same parameters $\omega_{aa}$ will be used on every timestep and similarly the sum $\omega_{ya}$ that governs the output predictions (Fig.21).

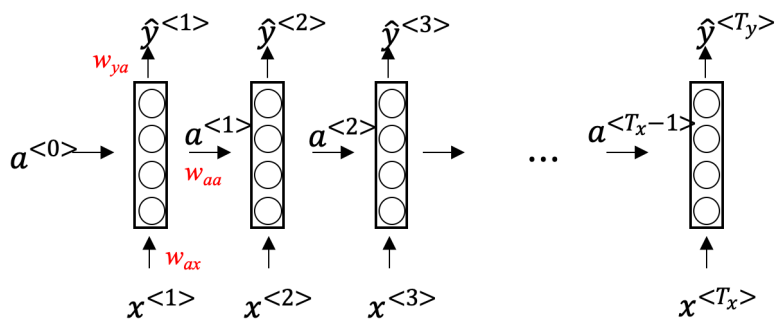


Figure 21: Training a recurrent neural network requires a set of parameters, which are adjusted during the training process (marked with red). We denote these parameters as $\omega$ and this is the same parameters that are used for every time step.

In this recurrent neural network, what this means is that when making the prediction for $y^n$, it gets the information not only from $x^n$ but also the information from $x^{n-1}$ and $x^{n+1}$ because the information on $x^{n-1}$ can help to predict $y^n$. One limitation of this particular neural network structure is that the prediction at a certain time uses inputs or uses information from the inputs earlier in the sequence but not information later in the sequence.

Now, let's define the steps of the neural network more formally. First, we initialise the activation $a^0$ with a vector of zeros:

$$a_0 = \overrightarrow{0} \tag{14}$$

On step one, we calculate activation $a_1$ and output $\hat{y}_1$ of the RNN:

$$a_1 = g(\omega_{aa} a_0 + \omega_{ax} x^1 + b_a), \tag{15}$$

where the model output is computed as

$$\hat{y}^1 = h(\omega_{ya} a^1 + b_y), \tag{16}$$

where $g$ and $h$ are activation functions, such as hyperbolic tangent, rectified linear unit (ReLu) or sigmoid.

Step $t$, therefore, can be described as following:

$$a^t = g(\omega_{aa} a_{t-1} + \omega_{ax} x^t + b_a), \tag{17}$$

and

$$\hat{y}^t = h(\omega_{ya} a^t + b_y). \tag{18}$$

Sometimes, visualisation of this process is slightly easier to understand. Fig.22 depicts the calculations, made in a single cell, in details.

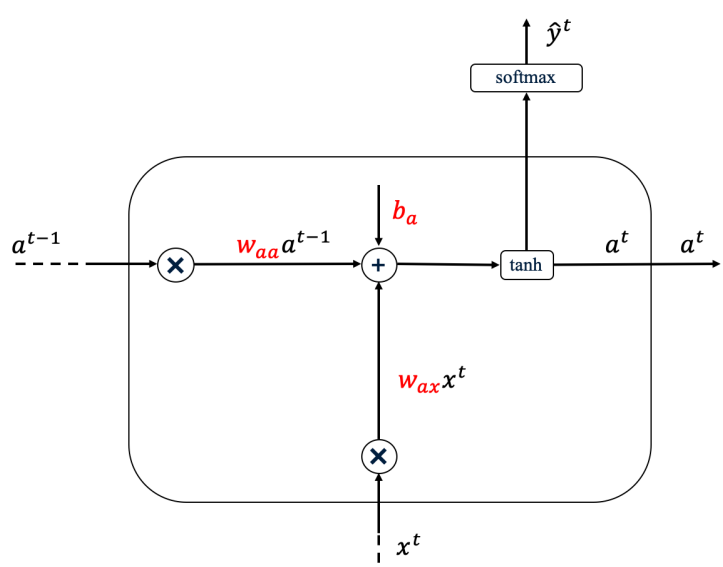


Figure 22: One "cell" of a recurrent neural network

Training of RNNs is also performed with a backpropagation algorithm, slightly modified for this purpose.

However, an RNN, described above, is the simplest version of a language model. Here, we will not go into greater details, but we encourage you to explore long-short term memory networks (or LSTMs). The best source to start from is Chris Olah's blog.

*2.2.2. Transformer models*

In the last few years, RNNs and LSTMs are quickly loosing popularity: they are unstable, difficult to train and have a limited ability to learn. The new version of language models, called transformer models, was introduced in 2018 and are currently replacing sequential approach.

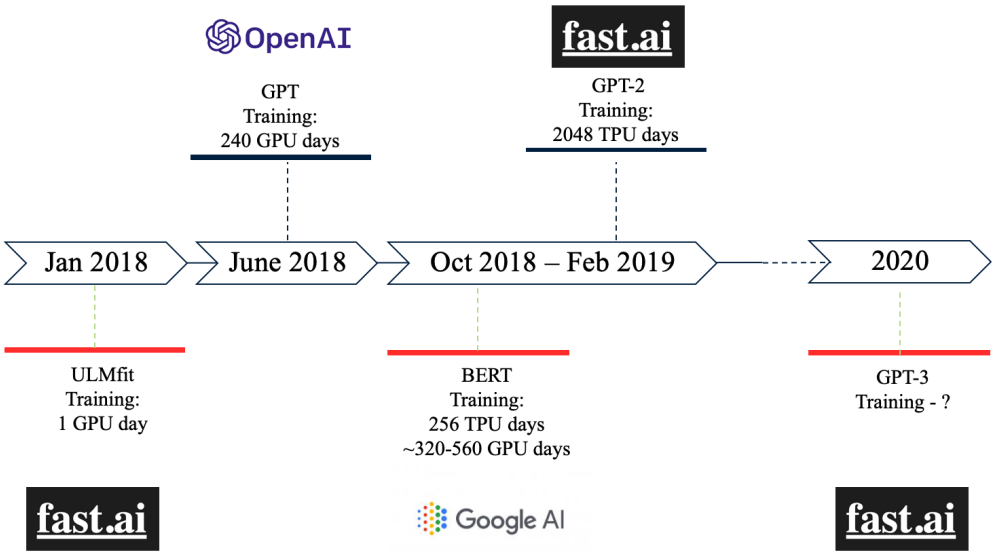


Figure 23: Evolution of transformer models.

Transformers were developed to solve the problem of neural machine translation. That means any task that transforms an input sequence to an output sequence. However, with small modifications, they can be allied to a variety of tasks (that's why they are called transformers: once trained, the can be easily modified for other problems).

Transformers try to solve the problem by using convolutional neural networks together with attention models. Attention boosts the speed of how fast the model can translate from one sequence to another. A great lecture on details of the transformer models and how they work you can find here.

There are a few features of transformer models that need to be mentioned. First, they are very big and difficult to train. Fig.23 shows the evolution of transformer models. Note that the first transformer were quite modest training-wise in comparison to the recent ones. Second important thing we need to mention, is that you cannot and do not need to train a transformer model yourself: the beauty of it is in the fact that you can use the pre-trained one for the majority of language tasks.

### *2.3. Generative adversarial networks*

Generative adversarial networks (GANs) are powerful models that learn to produce realistic objects, which are difficult to distinguish from existing real ones. They are particularly well known for their ability to produce realistic human faces (Fig.24).

Style GAN 2

BigGAN

Figure 24: Outputs of two well-known GANs: StyleGAN2 and BigGAN (in ML, developers of particularly successful models give them names!) Note that in the top row, one of the images has a defect, which s very typical for GANs. You won't see these defects online too often, so I have left it here on purpose so you can see what the actual output of this model looks like.

A GAN consists of two neural networks: a generator and a discriminator. It learns by making a generator and a discriminator compete against each other. The generator learns to generate fake images that look real, to fool the discriminator. And the discriminator learns to distinguish between what's

real and what's fake. Both models learn from their competition with each other, until the examples produced by the generator are good enough to fool the discriminator.

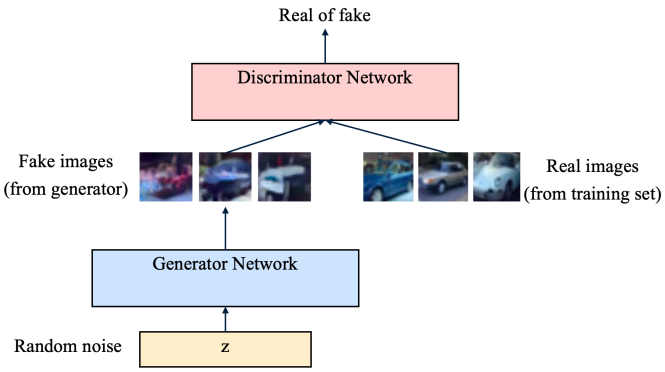


Figure 25: A GAN consists of a generator and a discriminator. Generator network tries to fool the discriminator by generating real-looking images. Discriminator network tries to distinguish between real and fake images. An input to the generator is random noise.

## 3. AI deployment

Deployment of AI can be considered as a new programming paradigm: Andrej Karpathy Software 2.0. In contrast to classis software, which consists of explicit instructions to the computer written by a programmer, Software 2.0 can be written in much more abstract, human unfriendly language, such as the weights of a neural network. No human is involved in writing this code because there are a lot of weights (typical networks might have millions), and coding directly in weights is really hard.

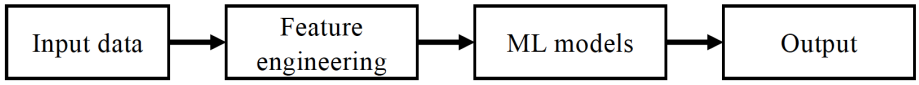


Figure 26: AI development pipeline.

### *3.1. Data component*

Feature engineering is an important part of AI development process. It allows to create meaningful inputs to machine learning models. Feature engineering is a complex process, it requires data formatting, data normalisation, filling in the gaps with missing data and in some cases creating inner representation of the data to make meaningful for ML models or reducing dimensionality of the input data.

### *3.2. ML for feature engineering*

Feature selection is the process of reducing the number of input variables when developing a predictive model. It is desirable to reduce the number of input variables to both reduce the computational cost of modelling and, in some cases, to improve the performance of the model. Statistical-based feature selection methods involve evaluating the relationship between each input variable and the target variable using statistics and selecting those input variables that have the strongest relationship with the target variable.

**Feature selection** (FS) methods select a subset of input features from the dataset. Unsupervised FS methods do not use the target variable (e.g. remove redundant variables), for example correlation techniques. In supervised

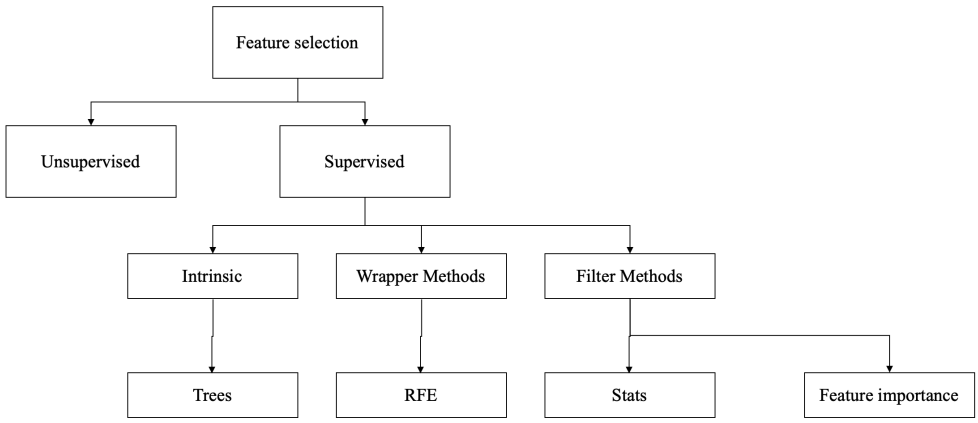


Figure 27: Summary of feature selection methods.

FS, we use the target variable (e.g. remove irrelevant variables). Wrapper methods perform search for well-performing subsets of features (e.g. RFE, Recursive Feature Elimination for Feature Selection). Filter methods select subsets of features based on their relationship with the target (e.g. statistical methods and feature importance methods). Intrinsic methods are algorithms that perform automatic feature selection during training, such as decision trees. There is another large group of methods (dimensionality reduction), which project input data into a lower-dimensional feature space.

*3.2.1. Pearson correlation*

It is common to use correlation type statistical measures between input and output variables as the basis for filter feature selection. The choice of statistical method depends on the type of input and output variables. In case where both input and output are numerical, Pearson correlation coefficient is used. Pearson's correlation coefficient, when applied to a sample, is referred to as the sample correlation coefficient or the sample Pearson correlation coefficient and described by the following equation:

$$r_{xy} = \frac{n(\sum xy) - (\sum x)(\sum y)}{\sqrt{[n\sum x^2 - (\sum x)^2][n\sum y^2 - (\sum y)^2]}} \tag{19}$$

where we have paired data $\{(x_1, y_1), ..., (x_n, y_n)\}$, consisting of $n$ pairs.

*3.2.2. Principal components analysis*

Principal components analysis (PCA) is one of a family of techniques for taking high-dimensional data, and using the dependencies between the variables to represent it in a more tractable, lower-dimensional form, without losing too much information. PCA is one of the simplest and most robust ways of doing such dimensionality reduction.

The underlying principle is the following: PCA tries to find a lower dimensional surface (in 3D space it will be 2 vectors, defining a plane, see Fig.28), onto which to project the data so that the sum of squares of the distances between the data point and the projection is minimised. These distances are called the projection error. This process requires mean normalisation and feature scaling.

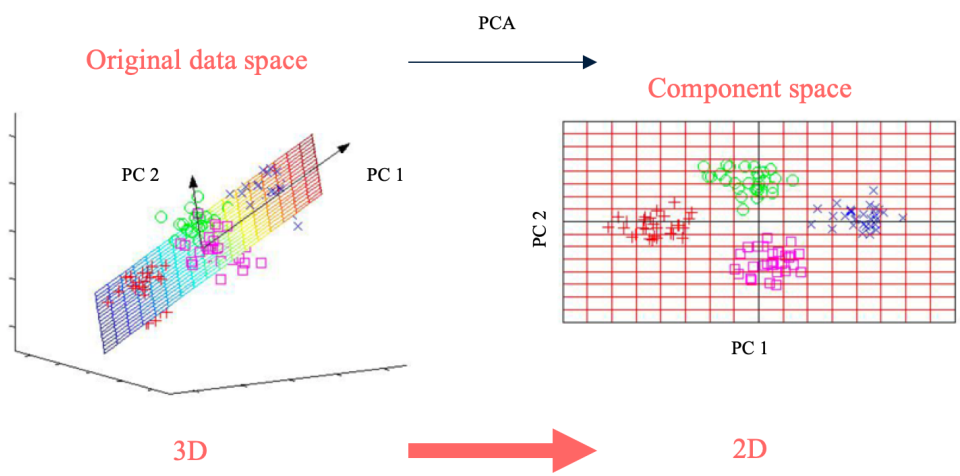


Figure 28: Principal components analysis tries to find a lower dimensional surface: it projects 3D space into 2D space. A plane is defined by 2 vectors: PC1 and PC2.

Formally, to reduce the dimensionality from $n$-dimensional space to $k$-dimensional space, we need to find a vector $u^{(1)}, ..., u^{(n)} \in \mathbb{R}^n$ onto which to project the data to minimise the projection error.

First, the input features are scaled (mean normalisation). For the training set $x^{(1)}, ..., x^{(n)}$, we first find the mean:

$$\mu_j = \frac{1}{m} \sum_{i=1}^{n} x_j^i \tag{20}$$

and replace each $x_j^i$ with $x_j - \mu_j$. After this, we scale features to have comparable range of values:

$$x_j^i \leftarrow \frac{x_j - \mu_j}{\sigma_j}, \tag{21}$$

where $\mu_j$ and $\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{n}(x_j^i - \mu_j)^2$ are the mean and the variance of feature $j$.

The next step is to compute the covariance matrix:

$$\Sigma = \frac{1}{m}\sum_{i=1}^{n}(x^{(i)})(x^{(i)})^T \tag{22}$$

More generally, if we wish to project our data into a $k$-dimensional subspace ($k < n$), we should choose $u_1, ..., u_k$ to be the top k eigenvectors of $\Sigma$. Then, to represent $x(i)$ in this basis, we need only compute the corresponding vector $y_i$:

$$y(i) = \begin{bmatrix} u_1^T x(i) \\ u_2^T x(i) \\ ... \\ u_k^T x(i) \end{bmatrix} \in \mathbb{R}^k. \tag{23}$$

Thus, whereas $x^{(i)} \in \mathbb{R}^k$, the vector $y^{(i)}$ now gives a lower, $k$-dimensional, approximation/representation for $x^{(i)}$. PCA is therefore also referred to as a dimensionality reduction algorithm. The vectors $u^{(1)}, ..., u^{(k)}$ are called the first $k$ principal components of the data.

*3.3. Development component*

Software engineering in ML has some ML-specific components, such as using ML frameworks, training and retraining of models and hyperparameter tuning (Fig.29).

A Hyperparameter is a parameter whose value is used to control the learning process. For example, in training a neural network hyperparameters will be a number of hidden layers, number of hidden units in each layer to have, learning rates and activation functions for the different layers. When developing a new application, it's very difficult to correctly guess the right values for all of these, and for other hyperparameter choices, on the first attempt, so it requires a separate process to set everything correctly to obtain a best performing model possible.

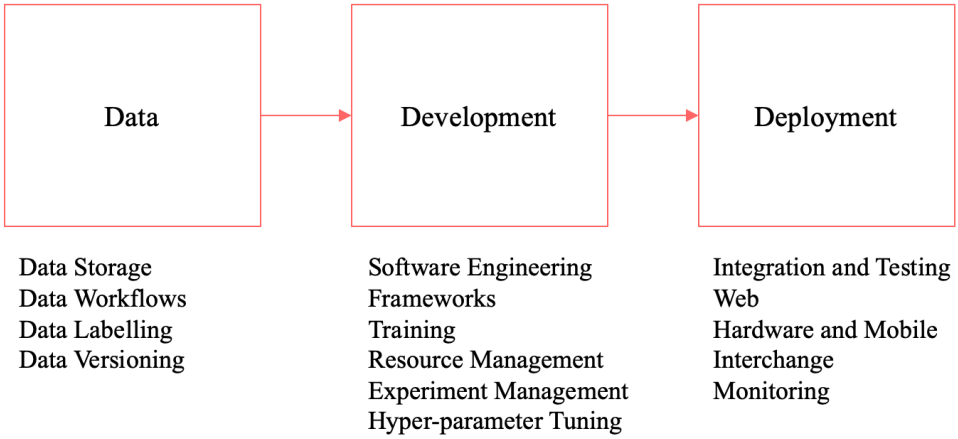


Figure 29: AI deployment.

### *3.4. Machine learning frameworks*

A Machine learning framework is a library or a tool which allows developers to more easily and quickly build machine learning models, without direct coding of the mathematics of the underlying algorithms. It provides a clear, concise way for defining machine learning models using a collection of pre-built, optimised tools. Typically ML frameworks are developed and continuously improved by research community. A few ML frameworks are originally developed by organisations, such as Google (TensorFlow) or Facebook (PyTorch), others are originated from large universities (e.g. Caffe is developed originally in Berkeley), whereas some are being developed by a community of researchers. All popular ML frameworks are open-source (free to use), maintained and updated by researchers.

There are multiple advantages in using ML frameworks: they are robust, well tested and simple to use. Using ML frameworks is a way to speed up the development process and avoid mistakes in implementing mathematical models.

According to Kaggle survey, the following ML frameworks are commonly used: Scikit-learn, TensorFlow, Keras, Xgboost and Pytorch (Fig.30).

#### *3.4.1. Scikit-learn*

Scikit-learn is a free software machine learning library for the Python pro-

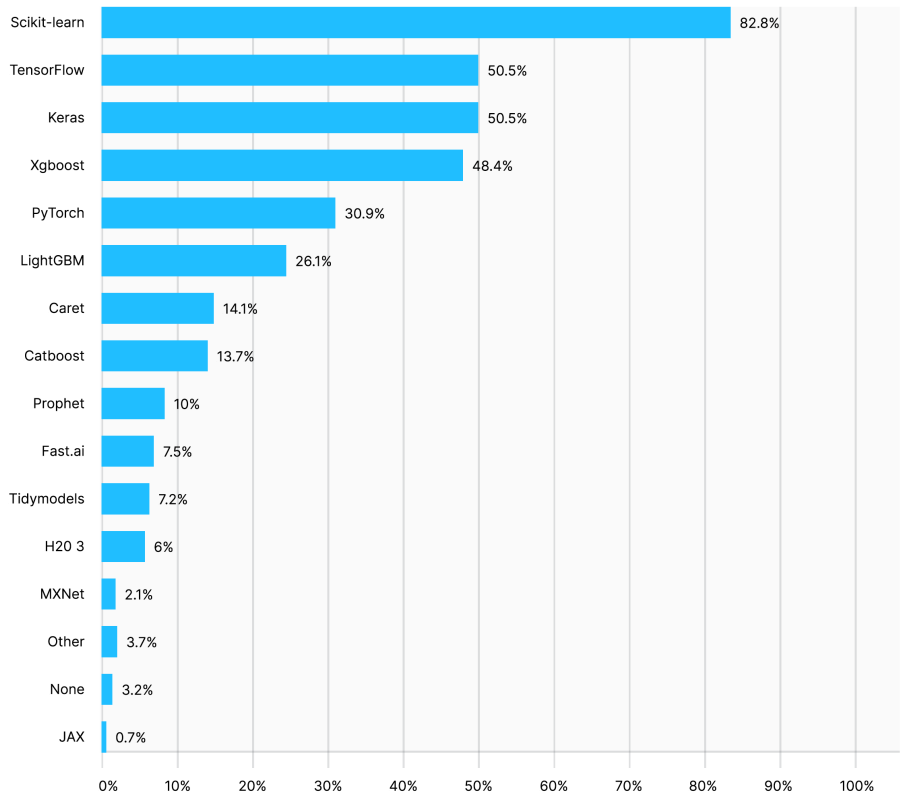


Figure 30: Machine learning framework usage.

gramming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting and k-means, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

It is the most basic ML framework and used to implement simple models, in comparison with TensorFlow and PyTorch, which are developed mostly for support of deep learning models.

### *3.4.2. TensorFlow*

TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

TensorFlow is a symbolic math library, and also used for machine learn-

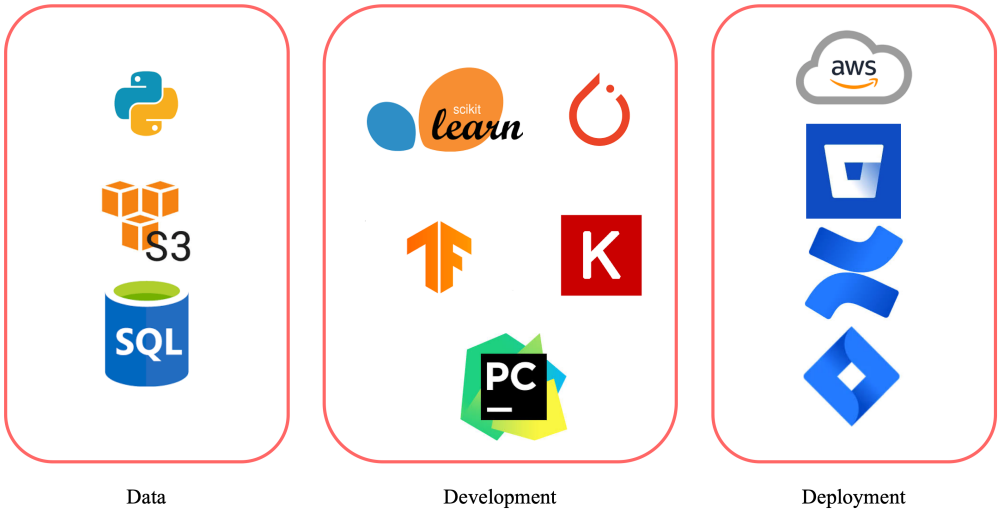


Figure 31: ML development tools (adapted from full stack deep learning course).

ing applications such as neural networks. It is used for both research and production at Google. TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache 2.0 open source license on November 9, 2015.

*3.4.3. Keras*

Keras is a higher-level framework than TensorFlow and is build on the top of TensorFlow or some other ML framework. In ML jargon, it "uses TensorFlow backend". It means it implements all TensorFlow functionality, but makes it easier to use the underlying framework. However, it is more restrictive in terms of flexibility of coding process and choice of some parameters.

*3.4.4. XGBoost*

XGboost framework implements machine learning algorithms under the gradient boosting framework. This framework has grown from a research project incubated in academia to the most widely used gradient boosting framework in production environment.

*3.4.5. PyTorch*

PyTorch is an open source machine learning library based on the Torch library, used for applications such as computer vision and natural language

processing, primarily developed by Facebook's AI Research lab (FAIR). It is free and open-source software released under the Modified BSD license. Although the Python interface is more polished and the primary focus of development, PyTorch also has a $C++$ interface.

### 4. Risks, biases and adversarial attacks

Adversarial examples are inputs to machine learning models that an attacker has intentionally designed to cause the model to make a mistake; they're like optical illusions for machines. Adversarial examples are a good aspect of security to work on because they represent a concrete problem in AI safety that can be addressed in the short term, and because fixing them is difficult enough that it requires a serious research effort.

A very typical example if adversarial attack is and image with added noise, which is invisible to humans but prevents a neural-network based computer vision system from recognising it correctly.

Other examples to add here: reliability, transparency, sensitivity, stability, sensitivity to adversarial attacks etc.

### 5. Build versus buy

Since this document is aimed for diploma, MBA and EMBA students, who do not specialise on AI, we felt that it is important to cover this subject.

In the last 30+ pages, we have discussed all sorts of AI and ML from the point of view of building and deploying it. However, in many cases building a ML tool is difficult, time consuming and simply not feasible if you have never done it before. If you need it for one-off single task (such as, for example, recognising images of workers in safety helmets or without and your main business is construction), you should consider just buying this service or outsourcing rather then hiring the whole development team to build it.

When you should rather buy a ready solution:

- when the application that you need is a "one-off" job and it is not going to change;
- when you do not have any data and it's unlikely you will ever have it;
- when you are particularly scared of ML and do not know when and how to start building your own AI, you can first try buying a service

and when you feel comfortable with it you should develop the next ML tool in-house.

We hope you enjoyed this short guide. If you have any suggestions on how it can be improved, please contact your course lecturer or an author of this document. And we wish you the best of luck in your AI journey!

## Glossary

**Adversarial examples** are inputs to machine learning models that an attacker has intentionally designed to cause the model to make a mistake; they're like optical illusions for machines. 38

**Artificial intelligence** a term coined by emeritus Stanford Professor John McCarthy in 1955, was defined by him as "the science and engineering of making intelligent machines". Much research has humans program machines to behave in a clever way, like playing chess, but, today, we emphasize machines that can learn, at least somewhat like human beings do. 4

**Backpropagation** is an algorithm that evaluates the gradient of the error function. It uses a local message passing scheme in which information is sent alternately forwards and backwards through the network and is known as error backpropagation. 17

**Computer vision** is an interdisciplinary scientific field that deals with how computers can gain high-level understanding from digital images or videos. From the perspective of engineering, it seeks to understand and automate tasks that the human visual system can do. From a machine learning perspective, computer vision is type of machine learning algorithms, which work with image and video data to understand it's content. 14

**Deep learning** is a sub-field of machine learning that works with more sophisticated algorithms that require very large amounts of data. Deep learning models, also known as neural networks, are the most complex algorithms in the field of AI. They are often the most difficult to train, the least interpretable and require the vast amounts of data. Due to the lack of transparency and difficulty to explain the result, they are often referred to as"black box" approach. However, they allow working with the most complex data and achieve the best performance on noisy data. 7

**Feature** is a piece of information used in the machine learning system as model input. 6

**Feature engineering** is an important part of modelling process. It allows to create meaningful inputs to machine learning models. Feature engineering is a complex process, it requires data formatting, data normalisation, filling in the gaps with missing data and in some cases creating inner representation of the data to make meaningful for machine learning models or reducing dimensionality of the input data. 31

**Feature selection** methods select a subset of input features from the dataset. 31

**Generative adversarial networks** are powerful models that learn to produce realistic objects. A GAN consists of two neural networks: a generator and a discriminator. It learns by making a generator and a discriminator compete against each other. The generator learns to generate fake images that look real, to fool the discriminator. And the discriminator learns to distinguish between what's real and what's fake. Both models learn from their competition with each other, until the examples produced by the generator are good enough to fool the discriminator. 29

**Hyperparameter** is a parameter whose value is used to control the learning process. For example, in training a neural network hyperparameters will be a number of hidden layers, number of hidden units in each layer to have, learning rates and activation functions for the different layers. 34

**Interpretability** Interpretability is the degree to which a human can understand the cause of a decision. In machine learning, interpretability is the degree to which a human can consistently predict the model's result. The higher the interpretability of a machine learning model, the easier it is for someone to comprehend why certain decisions or predictions have been made. From: Miller, Tim. "Explanation in artificial intelligence: Insights from the social sciences." arXiv Preprint arXiv:1706.07269. (2017). Kim, Been, Rajiv Khanna, and Oluwasanmi O. Koyejo. "Examples are not enough, learn to criticize! Criticism for interpretability." Advances in Neural Information Processing Systems (2016). 13

**Machine learning** is the part of artificial intelligence studying how computer agents can improve their perception, knowledge, thinking, or actions based on experience or data. For this, machine learning draws from computer science, statistics, psychology, neuroscience, economics and control theory. 4

**Machine learning algorithm** learns from data automatically, given a large number of samples from a particular sub-field of AI as inputs. This is a class of AI models with high variability in purpose and structure, with the the learning process being the only similarity between them. 5

**Machine learning framework** is a library or a tool which allows developers to more easily and quickly build machine learning models, without direct coding of the mathematics of the underlying algorithms. It provides a clear, concise way for defining machine learning models using a collection of pre-built, optimised tools. Typically machine learning frameworks are developed and continuously improved by research community. 35

**Naive Bayes** is a type of machine learning classifier based on applying Bayes' theorem while assuming that the features in the input data are all independent (a strong, or "naive" assumption, which is where the name comes from). 5

**Natural language processing** is a subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, in particular how to program computers to process and analyse large amounts of natural language data. 22

**One-hot encoding** is a technique of converting a word or a categorical variable into a machine-readable feature (number). For example, in text one-hot encoding associates a unique integer index with every word in the vocabulary and then turns this integer index $i$ into a binary vector of size $N$ (the size of the vocabulary); the resulting vector is all zeros except for the $i$-th entry, which is 1. 23

**Overfitting** means fitting the training data too well. This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model. The problem is that these concepts

do not apply to new data and negatively impact the models ability to generalise. To avoid this problem, regularisation is widely applied in machine learning models. 10

**Parameter** is a part of machine learning model, which is modified slightly during the training process. 16

**Probabilistic modelling** is the application of the principles of statistics to data analysis. It was one of the earliest forms of machine learning, and it's still widely used to this day. 5

**Recurrent neural network** is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. This allows it to exhibit temporal dynamic behaviour.. 25

**Regularization** is a technique used for tuning the function by adding an additional penalty term in the error function. The additional term controls the excessively fluctuating function such that the coefficients don't take extreme values. 10

**Reinforcement Learning** is focused on goal-directed learning from interaction between a learning agent and its environment). 11

**Representation** is a way to structure the data so it will be easily accessible to a machine learning algorithm. 6

**Representation learning** is a use of machine learning to discover not only the mapping from representation to output but also the representation itself. 6

**Rule-based systems** make inference from the data according to the predefined rules (called "propositions" or simply "if-then" rules). For example, traditional medical diagnostic systems are rule-based. Rules in rule-bases systems are created manually, from expert opinions, or semi-automatically, from existing data combined with expert opinions. Although this part of artificial intelligence is the oldest one, only a small proportion of commercial systems works without some type of rules at least in one component of the system. 4

**Scalability** is related to a model's ability to scale quickly from 100 data points to a million without a significant increase in computational complexity. 13

**Supervised Learning** In supervised learning, we are given a data set and already know what our correct output should look like, having the idea that there is a relationship between the input and the output. Supervised learning problems are categorized into "regression" and "classification" problems. In a regression problem, we are trying to predict results within a continuous output, meaning that we are trying to map input variables to some continuous function. In a classification problem, we are instead trying to predict results in a discrete output. 8

**Tensor** is a mathematical object that generalizes scalars, vectors and matrices to higher dimensions (i.e. $n$-dimensional array). It is a term and set of techniques known in machine learning in the training and operation of deep learning models can be described in terms of tensors. 22

**Training and test sets** To evaluate the performance of a machine learning model, we need to run it on the previously unseen data. For this reason, we divide our dataset into two subsets: training set and test set. We train our model on the training set and we will evaluate it's performance on the test set. 9

**Unsupervised Learning** Unsupervised learning allows us to approach problems with little or no idea what our results should look like. We can derive structure from data where we don't necessarily know the effect of the variables. We can derive this structure by clustering the data based on relationships among the variables in the data. With unsupervised learning there is no feedback based on the prediction results. 11

**Validation set** is a small set of data, on which we evaluate model's performance. Usually used together with training and test sets. 10

**Word embeddings** are learned representations for text where words that have the same meaning have a similar representation. Each word is mapped to one vector and the vector values are learned in a way that

resembles a neural network, therefore the technique is often connected to the field of deep learning. 23