A guide to machine learning for biologists

Joe G. Greener[1,2], *Shaun M. Kandathil*[1,2], *Lewis Moffat*[1] *and David T. Jones*[1] ✉

Abstract | The expanding scale and inherent complexity of biological data have encouraged a growing use of machine learning in biology to build informative and predictive models of the underlying biological processes. All machine learning techniques fit models to data; however, the specific methods are quite varied and can at first glance seem bewildering. In this Review, we aim to provide readers with a gentle introduction to a few key machine learning techniques, including the most recently developed and widely used techniques involving deep neural networks. We describe how different techniques may be suited to specific types of biological data, and also discuss some best practices and points to consider when one is embarking on experiments involving machine learning. Some emerging directions in machine learning methodology are also discussed.

Deep learning
Machine learning methods based on neural networks. The adjective 'deep' refers to the use of many hidden layers in the network, two hidden layers as a minimum but usually many more than that. Deep learning is a subset of machine learning, and hence of artificial intelligence more broadly.

Artificial neural networks
A collection of connected nodes loosely representing neuron connectivity in a biological brain. Each node is part of a layer and represents a number calculated from the previous layer. The connections, or edges, allow a signal to flow from the input layer to the output layer via hidden layers.

[1]*Department of Computer Science, University College London, London, UK.*

[2]*These authors contributed equally: Joe G. Greener, Shaun M. Kandathil.*

✉*e-mail: d.t.jones@ucl.ac.uk*

https://doi.org/10.1038/s41580-021-00407-0

Humans make sense of the world around them by observing it, and learning to predict what might happen next. Consider a child learning to catch a ball: the child (usually) knows nothing about the physical laws that govern the motion of a thrown ball; however, by a process of observation, trial and error, the child adjusts his or her understanding of the ball's motion, and how to move his or her body, until he or she is able to catch it reliably. In other words, the child has learned how to catch the ball by building a sufficiently accurate and useful 'model' of the process, by repeatedly testing this model against the data and by making corrections to the model to make it better.

'Machine learning' refers broadly to the process of fitting predictive models to data or of identifying informative groupings within data. The field of machine learning essentially attempts to approximate or imitate humans' ability to recognize patterns, albeit in an objective manner, using computation. Machine learning is particularly useful when the dataset one wishes to analyse is too large (many individual data points) or too complex (contains a large number of features) for human analysis and/or when it is desired to automate the process of data analysis to establish a reproducible and time-efficient pipeline. Data from biological experiments frequently possess these properties; biological datasets have grown enormously in both size and complexity in the past few decades, and it is becoming increasingly important not only to have some practical means of making sense of this data abundance but also to have a sound understanding of the techniques that are used. Machine learning has been used in biology for a number of decades, but it has steadily grown in importance to the point where it is used in nearly every field of biology. However, only in the past few years has the field taken a more critical look at the available strategies and begun to assess which methods are most appropriate in different scenarios, or even whether they are appropriate at all.

This Review aims to inform biologists on how they can start to understand and use machine learning techniques. We do not intend to present a thorough literature review of articles using machine learning for biological problems[1], or to describe the detailed mathematics of various machine learning methods[2,3]. Instead, we focus on linking particular techniques to different types of biological data (similar reviews are available for specific biological disciplines; see, for example, REFS[4–11]). We also attempt to distil some best practices of how to practically go about the process of training and improving a model. The complexity of biological data presents pitfalls as well as opportunities for their analysis using machine learning techniques. To address these, we discuss the widespread issues that affect the validity of studies, with guidance on how to avoid them. The bulk of the Review is devoted to the description of a number of machine learning techniques, and in each case we provide examples of the appropriate application of the method and how to interpret the results. The methods discussed include traditional machine learning methods, as these are still the best choices in many cases, and deep learning with artificial neural networks, which are emerging as the most effective methods for many tasks. We finish by describing what the future holds for incorporating machine learning in data analysis pipelines in biology.

There are two goals when one is using machine learning in biology. The first is to make accurate predictions

where experimental data are lacking, and use these predictions to guide future research efforts. However, as scientists we seek to understand the world, and so the second goal is to use machine learning to further our understanding of biology. Throughout this guide we discuss how these two goals often come into conflict in machine learning, and how to extract understanding from models that are often treated as 'black boxes' because their inner workings are difficult to understand[12].

Key concepts

We first introduce a number of key concepts in machine learning. Where possible, we illustrate these concepts with examples taken from biological literature.

General terms. A dataset comprises a number of data points or instances, each of which can be thought of as a single observation from an experiment. Each data point is described by a (usually fixed) number of features. Examples of such features include length, time, concentration and gene expression level. A machine learning task is an objective specification for what we want a machine learning model to accomplish. For example, for an experiment investigating the expression of genes over time, we might want to predict the rate of conversion of a specific metabolite into another species. In this case, the features 'gene expression level' and 'time' could be termed input features or simply inputs for the model, and 'conversion rate' would be the desired output of the model; that is, the quantity we are interested in predicting. A model can have any number of input and output features. Features can be either continuous (taking continuous numerical values) or categorical (taking only discrete values). Quite often, categorical features are simply binary and are either true (1) or false (0).

Supervised and unsupervised learning. 'Supervised machine learning' refers to the fitting of a model to data (or a subset of data) that have been labelled — where there exists some ground truth property, which is usually experimentally measured or assigned by humans. Examples include protein secondary structure prediction[13] and prediction of genome accessibility to genome-regulatory factors[14]. In both cases, the ground truth is derived ultimately from laboratory observations, but often these raw data are preprocessed in some way. In the case of secondary structure, for example, the ground truth data are derived from analysing protein crystal structure data in the Protein Data Bank, and in the latter case, the ground truth comes from data derived from DNA-sequencing experiments. By contrast, unsupervised learning methods are able to identify patterns in unlabelled data, without the need to provide the system with the ground truth information in the form of predetermined labels, such as finding subsets of patients with similar expression levels in a gene expression study[15] or predicting mutation effects from gene sequence co-variation[16]. Sometimes the two approaches are combined in semi-supervised learning, where small amounts of labelled data are combined with large amounts of unlabelled data. This can improve performance in cases where labelled data are costly to obtain.

Classification, regression and clustering problems. When a problem involves assigning data points to a set of discrete categories (for example, 'cancerous' or 'not cancerous'), the problem is called a 'classification problem', and any algorithm that performs such classification can be said to be a classifier. By contrast, regression models output a continuous set of values, such as predicting the free energy change of folding after mutating a residue in a protein[17]. Continuous values can be thresholded or otherwise discretized, meaning that it is often possible to reformulate regression problems as classification problems. For example, the free energy change mentioned above can be binned into ranges of values that are favourable or unfavourable for protein stability. Clustering methods are used to predict groupings of similar data points in a dataset, and are usually based on some measure of similarity between data points. They are unsupervised methods that do not require that the examples in a dataset have labels. For example, in a gene expression study, clustering could find subsets of patients with similar gene expression.

Classes and labels. The discrete set of values returned by a classifier can be made to be mutually exclusive, in which case they are called 'classes'. Where these values need not be mutually exclusive, they are termed 'labels'. For example, a residue in a protein structure can be in only one of multiple secondary structure classes, but could simultaneously be assigned the non-exclusive labels of being α-helical and transmembrane. Classes and labels are usually represented by an encoding (for example, a one-hot encoding).

Loss or cost functions. The output or outputs of a machine learning model are never ideal and will diverge from the ground truth. The mathematical functions that measure this deviation or in more general terms that measure the amount of 'disagreement' between the obtained and ideal outputs are referred to as 'loss functions' or 'cost functions'. In supervised learning settings, the loss function would be a measure of deviation of the output relative to the ground truth output. Examples include mean squared error loss for regression problems and binary cross entropy for classification problems.

Parameters and hyperparameters. Models are essentially mathematical functions that operate on some set of input features and produce one or more output values or features. To be able to learn on training data, models contain adjustable parameters whose values can be changed over the training process to achieve the best performance of the model (see later). In a simple regression model, for example, each feature has a parameter that is multiplied by the feature value, and these are added together to make the prediction. Hyperparameters are adjustable values that are not considered part of the model itself in that they are not updated during training, but which still have an impact on the training of the model and its

Ground truth
The true value that the output of a machine learning model is compared with to train the model and test performance. These data usually come from experimental data (for example, accessibility of a region of DNA to transcription factors) or expert human annotation (for example healthy or pathological medical image).

Encoding
Any scheme for numerically representing (often categorical) data in a form suitable for use in a machine learning model. An encoding can be a fixed numerical representation (for example, one-hot or continuous encoding) or can be defined using parameters that are trained along with the rest of a model.

One-hot encoding
An encoding scheme that represents a fixed set of *n* categorical inputs using *n* unique *n*-dimensional vectors, each with one element set to 1 and the rest set to 0. For example, the set of three letters (A,B,C) could be represented by the three vectors [1,0,0], [0,1,0] and [0,0,1], respectively.

Mean squared error
A loss function that calculates the average squared difference between the predicted values and the ground truth. This function heavily penalizes outliers because it increases rapidly as the difference between a predicted value and the ground truth grows.

Binary cross entropy
The most common loss function for training a binary classifier; that is, for tasks aimed at answering a question with only two choices (such as cancer versus non-cancer); sometimes called 'log loss'.

performance. A common example of a hyperparameter is the learning rate, which controls the rate or speed with which the model's parameters are altered during training.

Training, validation and testing. Before being used to make predictions, models require training, which involves automatically adjusting the parameters of a model to improve its performance. In a supervised learning setting, this involves modifying the parameters so the model performs well on a training dataset, by minimizing the average value of the loss or cost function (described earlier). Usually, a separate validation dataset is used to monitor but not influence the training process so as to detect potential overfitting (see the next section). In unsupervised settings, a cost function is still minimized, although it does not operate on ground truth outputs. Once a model is trained, it can be tested on data not used for training. See BOX 1 for a guide to the overall process of training and how to split the data appropriately between training and testing sets. A flowchart to help the overall process is shown in FIG. 1, and some of the concepts in model training are shown in FIG. 2.

Box 1 | **Doing machine learning**

Here we outline the steps that should be taken when one is training a machine learning model. There is surprisingly little guidance available on the model selection and training process[146,147], with descriptions of the stepping stones and failed models rarely making it into published research articles. The first step, before touching any machine learning code, should be to fully understand the data (inputs) and prediction task (outputs) at hand. This means a biological understanding of the question, such as knowing the origin of the data and the sources of noise, and having an idea of how the output could theoretically be predicted from the input using biological principles. For example, it can be reasoned that different amino acids might have preferences for particular secondary structures in proteins, so it makes sense to predict secondary structure from amino acid frequencies at each position in a protein sequence. It is also important to know how the inputs and outputs are stored computationally. Are they normalized to prevent one feature having an unduly large influence on prediction? Are they encoded as binary variables or continuously? Are there duplicate entries? Are there missing data elements?

Next, the data should be split to allow training, validation and testing. There are a number of ways to do this, two of which are shown in FIG. 2a. The training set is used to directly update the parameters of the model being trained. The validation set, usually around 10% of the available data, is used to monitor training, select hyperparameters and prevent the model overfitting to the training data. Often *k*-fold cross-validation is used: the training set is split into *k* evenly sized partitions (for example, five or ten) to form *k* different training and validation sets, and the performance is compared across each partition to select the best hyperparameters. The test set, sometimes called the 'hold-out set', typically also around 10% of the available data, is used to assess the performance of the model on data not used for training or validation (that is, estimate its expected real-world performance). The test set should be used only once, at the very end of the study, or as infrequently as possible[27,38] to avoid tuning the model to fit the test set. See the section Data leakage for issues to consider when making a fair test set.

The next step is model selection, which depends on the nature of the data and the prediction task, and is summarized in FIG. 1. The training set is used to train the model following best practices of the software framework being used. Most methods have a handful of hyperparameters that need to be tuned to achieve the best performance. This can be done using random search or grid search, and can be combined with *k*-fold cross-validation as outlined above[27]. Model ensembling should be considered, where the outputs of a number of similar models are simply averaged to give a relatively reliable way to boost overall accuracy of the modelling task. Finally, the accuracy of the model on the test set (see above) should be assessed.

Overfitting and underfitting. The purpose of fitting a model to training data is to capture the 'true' relationship between the variables in the data, such that the model has predictive power on unseen (non-training) data. Models that are either overfitted or underfitted will produce poor predictions on data not in the training set (FIG. 2d). An overfitted model will produce excellent results on data in the training set (usually as a result of having too many parameters), but will produce poor results on unseen data. The overfitted model in FIG. 2d passes exactly through every training point, and so its prediction error on the training set will be zero. However, it is evident that this model has 'memorized' the training data and is unlikely to produce good results on unseen data. By contrast, an underfitted model fails to adequately capture the relationships between the variables in the data. This could be due to an incorrect choice of model type, incomplete or incorrect assumptions about the data, too few parameters in the model and/or an incomplete training process. The underfitted model depicted in FIG. 2d is inadequate for the data it is trying to fit; in this case it is evident that the variables have a non-linear relationship, which cannot be adequately described with a simple linear model and so a non-linear model would be more appropriate.

Inductive bias and the bias–variance trade-off. The 'inductive bias' of a model refers to the set of assumptions made in the learning algorithm that leads it to favour a particular solution to a learning problem over others. It can be thought of as the model's preference for a particular type of solution to a learning problem over others. This preference is often programmed into the model using its specific mathematical form and/or by using a particular loss function. For example, the inductive bias of recurrent neural networks (RNNs; discussed later) is that there are sequential dependencies in the input data such as the concentration of a metabolite over time. This dependence is explicitly accounted for in the mathematical form of an RNN. Different inductive biases in different model types make them more suitable and usually better performing for specific types of data. Another important concept is the trade-off between bias and variance. A model with a high bias can be said to have stronger constraints on the trained model, whereas a model with low bias makes fewer assumptions about the property being modelled, and can, in theory, model a wide variety of function types. The variance of a model describes how much the trained model changes in response to training it on different training datasets. In general, we desire models with very low bias and low variance, although these objectives are often in conflict as a model with low bias will often learn different signals on different training sets. Controlling the bias–variance trade-off is key to avoiding overfitting or underfitting.

Traditional machine learning

We now discuss several key machine learning methods, with an emphasis on their particular strengths and weaknesses. A comparison of different machine learning approaches is shown in TABLE 1. We begin with a discussion of methods not based on neural networks,

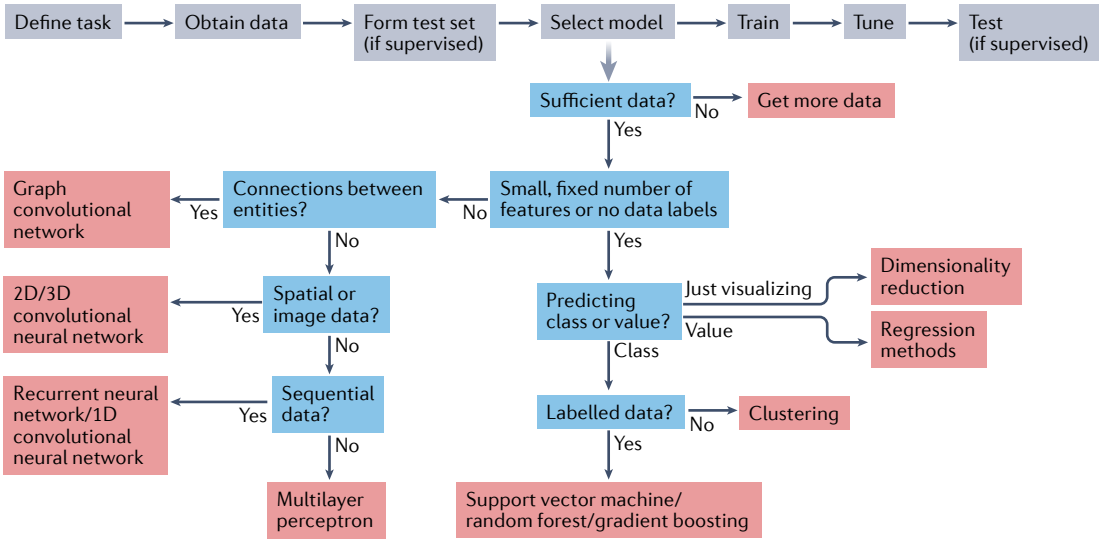

Fig. 1 | **Choosing and training a machine learning method.** The overall procedure for training a machine learning method is shown along the top. A decision tree to assist researchers in selecting a model is given below. This flowchart is intended to be used as a visual guide linking the concepts outlined in this Review. However, a simple overview such as this cannot cover every case. For example, the number of data points required for machine learning to become applicable depends on the number of features available for each data point, with more features requiring more data points, and also depends on the model being used. There are also deep learning models that work on unlabelled data.

sometimes called 'traditional machine learning'. FIGURE 3 shows some of the methods of traditional machine learning. Various software packages can be used to train such models, including scikit-learn in Python[18], caret in R[19] and MLJ in Julia[20].

When one is developing machine learning methods for use with biological data, traditional machine learning should generally be seen as the first area to explore in finding the most appropriate method for a given task. Deep learning can be a powerful tool, and is undeniably trendy currently. However, it is still limited in the application areas in which it excels: when large amounts of data are available (for example, millions of data points); when each data point has many features; and when the features are highly structured (the features have clear relationships with one another, such as adjacent pixels in images)[21]. Data such as DNA, RNA and protein sequences[22,23] and microscopy images[24,25] are examples of biological data where these requirements can be met and deep learning has been successfully applied. However, the requirement for large amounts of data can make deep learning a poor choice even when the other two requirements are met.

Traditional methods, in comparison to deep learning, are much faster to develop and test on a given problem. Developing the architecture of a deep neural network and then training it can be a time-consuming and computationally expensive task to undertake[26] compared with traditional models such as support vector machines (SVMs) and random forests[27]. Although some approaches exist, with deep neural networks it is still not trivial to estimate feature importance[28] (that is, how important each feature is for contributing to the prediction) or the confidence of predictions of the model[1,28,29], both of which are often essential in biological settings. Even if deep learning appears technically feasible for a particular biological prediction task, it is often still prudent to train a traditional method to compare it against a neural network-based model, if possible[30].

Traditional methods typically expect that each example in the dataset has the same number of features, so this is not always possible. An obvious biological example of this is when protein, RNA or DNA sequences are being used and each example has a different length. To use traditional methods with these data, the data can be altered so they are all the same size using simple techniques such as padding and windowing. 'Padding' means taking each example and adding additional values containing zero until it is the same size as the largest example in the dataset. By contrast, windowing shortens individual examples to a given size (for example, using only the first 100 residues of each protein in a dataset of protein sequences with lengths ranging from 100 upwards).

Use of classification and regression models. For regression problems such as those shown in FIG. 3a, ridge regression (linear regression with a regularization term) is often a good starting point for developing a model, as it can provide a fast and well-understood benchmark for a given task. Other variants of linear regression such as LASSO regression[31] and elastic net regression[32] are also worth considering when there is a desire for a model to rely on a minimal number of features within the available data. Unfortunately, the relationships between features in the data are often non-linear, and so use of a model such as an SVM is often a more appropriate choice for these cases[33]. SVMs are a powerful type of regression and classification model that uses kernel functions to transform a non-separable problem into a separable problem that is easier to solve. SVMs can be used to perform both linear regression and non-linear regression depending on the kernel function used[34–37]. A good approach to developing

Linear regression
A model that assumes that the output can be calculated from a linear combination of inputs; that is, each input feature is multiplied by a single parameter and these values are added. It is easy to interpret how these models make their predictions.

Kernel functions
Transformations applied to each data point to map the original points into a space in which they become separable with respect to their class.

Non-linear regression
A model where the output is calculated from a non-linear combination of inputs; that is, the input features can be combined during prediction using operations such as multiplication. These models can describe more complex phenomena than linear regression.

k nearest neighbours
A classification approach where a data point is classified on the basis of the known (ground truth) classes of the *k* most similar points in the training set using a majority voting rule. *k* is a parameter that can be tuned. Can also be used for regression by averaging the property value over the *k* nearest neighbours.

a model is to train a linear SVM and an SVM with a radial basis function kernel (a general-purpose non-linear type of SVM) to quantify what gain, if any, can be had from a non-linear model. Non-linear approaches can provide more powerful models but at the cost of easy interpretation of which features are influencing the model, a trade-off mentioned in the introduction.

Many of the models that are commonly used in regression are also used for classification. Training a linear SVM and an SVM with a radial basis function kernel is also a good default starting point for a classification task. An additional method that can be tried is *k* nearest neighbours classification[38]. Being one of the simplest classification methods, *k* nearest neighbours classification provides a useful baseline performance marker against which other more complex models, such as SVMs, can be compared. Another class of robust non-linear methods is ensemble-based models such as random forests[39] and XGBoost[40,41]. Both methods are powerful non-linear models that have the added benefits of providing feature importance estimates and often requiring minimal hyperparameter tuning. Due to the assignment of feature importance values and the decision tree structure, these models are a good choice if understanding which features contributed the most to a prediction is essential for biological understanding.

For both classification and regression, the many available models tend to have a bewildering variety of flavours and variants. Trying to predict how well suited a particular method will be to a particular problem a priori can be deceptive, and instead taking an empirical, trial-and-error approach to finding the best model is generally the most prudent approach. With modern machine learning suites such as scikit-learn[18], changing between these model variants often requires changing just one line of code, so a good overall strategy for selecting the best method is to train and optimize a variety of the aforementioned methods and choose the one with the best performance on the validation set before finally comparing their performance on a separate test set.

Use of clustering models. The use of clustering algorithms (FIG. 3e) is pervasive within biology[42,43]. *k*-means is a strong general purpose approach to clustering that, like many other clustering algorithms, requires the number of clusters to be set as a hyperparameter[44]. DBSCAN is an alternative method that does not require the number of clusters to be predefined, but has the trade-off that other hyperparameters have to be set[45]. Dimensionality reduction can also be performed before clustering to improve performance for datasets with a large number of features.

Dimensionality reduction. Dimensionality reduction techniques are used to transform data with a large number of attributes (or dimensions) into a lower-dimensional form while preserving the different relationships between the data points as much as possible.

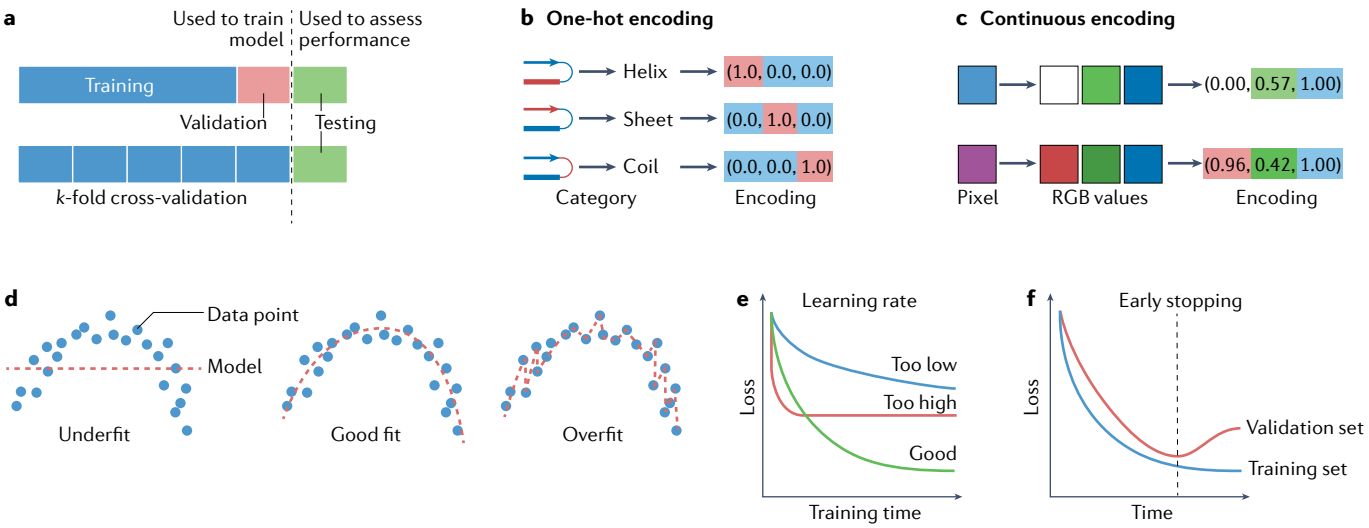

Fig. 2 | **Training machine learning methods. a** | Available data are often split into training, validation and test sets. The training set is directly used to train the model, the validation set is used to monitor training and the test set is used to assess the performance of the model. *k*-fold cross-validation with a test set can also be used. **b** | One-hot encoding is a common approach for representing categorical inputs where a single choice is permitted from a number of possibilities, in this case three possible protein secondary structure classes. The result of the encoding is a vector with three numbers, all equal to 0 except the occupied class, which is set to 1. This vector is used by the machine learning model. **c** | Continuous encoding represents numerical inputs, in this case the red, green and blue (RGB) values of a pixel in an image. Again the result is a vector with three numbers, corresponding to the amount of red, green and blue in the pixel. **d** | Failing to learn the underlying relationship between the variables is called 'underfitting', whereas learning the noise in the training data is called 'overfitting'. Underfitting can be caused by using a model without sufficient complexity to describe the signal. Overfitting can be caused by using a model with too many parameters or by continuing training after it has learned the true relationship between the variables. **e** | The learning rate of the model determines how quickly learned parameters are adjusted when training a neural network or some traditional methods such as gradient boosting. A low learning rate can lead to slow training, which is time-consuming and requires considerable computing power. By contrast, a high learning rate can lead to quick convergence on a non-optimal solution and poor performance of the model. **f** | Early stopping is the process of terminating training at the point where the loss function on the validation set starts to increase, even if the loss function on the training set is still decreasing. Use of early stopping can prevent overfitting.

Table 1 | **Comparison of different machine learning methods**

Method	Type of data	Example applications	Advantages	Disadvantages
Ridge (and LASSO/elastic) regression	Labelled Fixed number of features	Protein-variant effect prediction[122] Chemical/biochemical reaction kinetics[123]	Easy to interpret Easy to train Good benchmark	Cannot learn complex feature relationships Overfits with a large number of features
Support vector machine	Labelled Fixed number of features	Protein function prediction[124] Transmembrane-protein topology prediction[125]	Can perform both linear and non-linear classification and regression	Scaling to large datasets is often difficult
Random forest	Labelled Fixed number of features	Prediction of disease-associated genome mutations[126] Scoring of protein–ligand interactions[39]	Learns how important each feature is to the prediction Individual decision trees are human readable, allowing interpretation of how a decision is made Less sensitive to feature scaling and normalization so easier to train and tune	Less appropriate for regression Many decision trees are hard to interpret
Gradient boosting (for example, XGBoost)	Labelled Fixed number of features	Gene expression profiling[127]	Learns how important each feature is to the prediction Decision trees are human-readable, allowing interpretation of how a decision is made Less sensitive to feature scaling and normalization so easier to train and tune	Can struggle to learn underlying signal if noise is present Less appropriate for regression
Clustering	Unlabelled Fixed number of features	Differential gene expression analysis[15] Model selection in protein structure prediction[128]	For low-dimensional data, good clustering is easily identifiable Cluster validation metrics are available to assess performance	Scaling to large datasets is difficult for some methods Noisy datasets sometimes yield contradictory results
Dimensionality reduction	Unlabelled Large and fixed number of features	Single-cell transcriptomics[49] Analysis of molecular-dynamics trajectories[129]	Provides visual representation of data Goodness-of-fit evaluations usually available to assess performance	Hard to preserve both global and local differences in data Scaling to large numbers of samples is difficult for some methods
Multilayer perceptron	Labelled Fixed number of features	Protein secondary structure prediction[13] Drug toxicity prediction[54]	Can fit datasets with fewer layers than architectures such as convolutional neural networks, making it easier and faster to train	Easy to overfit Large number of parameters Hard to interpret
Convolutional neural network	Spatial data arranged in a grid; for example, 2D image (pixels) or 3D volumes (voxels) Allows variable input size	Protein residue–residue contact and distance prediction[23] Medical image recognition[24]	Variable input size Learns patterns irrespective of location in input	Receptive field, the amount of the input that is considered when predicting the output for each pixel, can be limited Hard to train deeper architectures that use many layers to increase the receptive field and make more complex predictions
Recurrent neural network	Sequential data (for example, biological sequences or time-series data) Allows variable input size	Protein engineering[68] Predicting clinical events[66]	Variable input size Sequences are found in many areas of biology	Long training times High computing memory requirements
Graph convolutional network	Data characterized by connections between entities (spatial, interaction or association) Allows variable input size	Predicting drug properties[77] Interpreting molecular structures[73,74] Knowledge extraction[130]	Variable graph sizes supported, which is important because most graphs in biology have variable size Learns patterns by following graph connectivity so predictor uses most relevant associations	High computing memory requirements for large, densely connected graphs Hard to train deeper architectures

Table 1 (cont.) | **Comparison of different machine learning methods**

Method	Type of data	Example applications	Advantages	Disadvantages
Autoencoders	Labelled or unlabelled data Fixed or variable input size depending on architecture	Protein and gene engineering[82] Prediction of DNA methylation[81] Neural population dynamics[131]	Latent space provides low-dimensional representation that can be used to visualize input data Can generate new samples, which is useful in areas such as protein design	Latent space specific to data in training set and may not be appropriate to other datasets Testing newly generated samples often requires wet laboratory experiments

Each method has types of data and applications to which it is best suited, along with advantages and disadvantages when compared with other methods.

For example, data points that are similar (for example, two homologous protein sequences) should also be similar in their lower-dimensional form, whereas dissimilar data points (for example, unrelated protein sequences) should remain dissimilar[46,47]. Two or three dimensions are often chosen to allow visualization of the data on a set of axes, although larger numbers of dimensions have uses in machine learning too. These techniques comprise both linear and non-linear transformations of the data. Examples common in biology include principal component analysis (PCA) as shown in FIG. 3d, Uniform Manifold Approximation and Projection (UMAP) and *t*-distributed stochastic neighbour embedding (*t*-SNE)[48]. The technique to use depends on the situation: PCA retains global relationships between data points and is interpretable because each component is a linear combination of input features, meaning it is easy to understand which features give rise to variety in the data. *t*-SNE more strongly preserves local relationships between data points and is a flexible method that can reveal structure in complex datasets. Applications include single-cell transcriptomics for *t*-SNE[49] and molecular dynamics trajectory analysis for principal component analysis.

Artificial neural networks

Artificial neural network models get their name from the fact that the form of the mathematical model that is being fit is inspired by the connectivity and behaviour of neurons in the brain and was originally designed to learn about brain function[50]. However, the neural networks in common use in data science are obsolete as brain models, and are now just machine learning models that can offer state-of-the-art performance in certain applications. Interest in neural network models has grown in recent decades owing to rapid advances in the architectures and training of deep neural networks[26]. In this section, we describe basic neural networks, as well as varieties that are widely used in biological studies. Some of these are shown in FIG. 4.

Basic principles of neural networks. A key property of neural networks is that they are universal function approximators, which means that, with very few assumptions, a correctly configured neural network can approximate any mathematical function to an arbitrary level of accuracy. In other words, if any process (biological or otherwise) can be thought of as some function of a set of variables, then that process can be modelled to any arbitrary degree of accuracy, governed by just the size or complexity of the model. The above definition of universal approximation is not mathematically rigorous, but does highlight one reason why interest in neural networks has persisted for decades. However, this guarantee does not provide a way of finding the optimal parameters of a neural network model that will produce the best approximation for a given dataset. There is also no guarantee that the model will provide accurate predictions for new data[51].

Artificial neurons are the building blocks of all neural network models. An artificial neuron is simply a mathematical function that maps (converts) inputs to outputs in a specific way. A single artificial neuron takes in any number of input values, applies a specific mathematical function to them and returns an output value. The function used is usually represented as

$$y = \sigma\left(\sum_{i=1}^{n} (w_i x_i) + b\right), \quad (1)$$

where x_i represents a single input variable or feature (there are n such inputs), w_i represents a learnable weight for that input, b represents a learnable bias term and σ represents a non-linear activation function that takes a single input and returns a single output. To create a network, artificial neurons are arranged in layers, with the output of one layer being the input to the next. The nodes of the network can be thought of as holding the y values from the above equation, which become the x values for the next layer. We describe various approaches for arranging artificial neurons in the following subsections, which are called 'neural network architectures'. It is also common to combine the different architecture types; for example, in a convolutional neural network (CNN) used for classification, fully connected layers are usually used to produce the final classification output.

Multilayer perceptrons. The most basic layout of a neural network model is that of layers of artificial neurons arranged in a fully connected fashion, as shown in FIG. 4a. In this layout, a fixed number of 'input neurons' represent the input feature values calculated from the data that are fed to the network, and each connection between a pair of neurons represents one trainable weight parameter. These weights are the main adjustable parameters in a neural network, and optimizing these weights is what is meant by neural network training. At the other end of the network, a number of output neurons represent the final output values from the network. Such a network, when correctly configured, can be used to make complex, hierarchical decisions about the input, as each neuron in a given layer receives inputs from all neurons in the previous layer. Layers of neurons in this simple

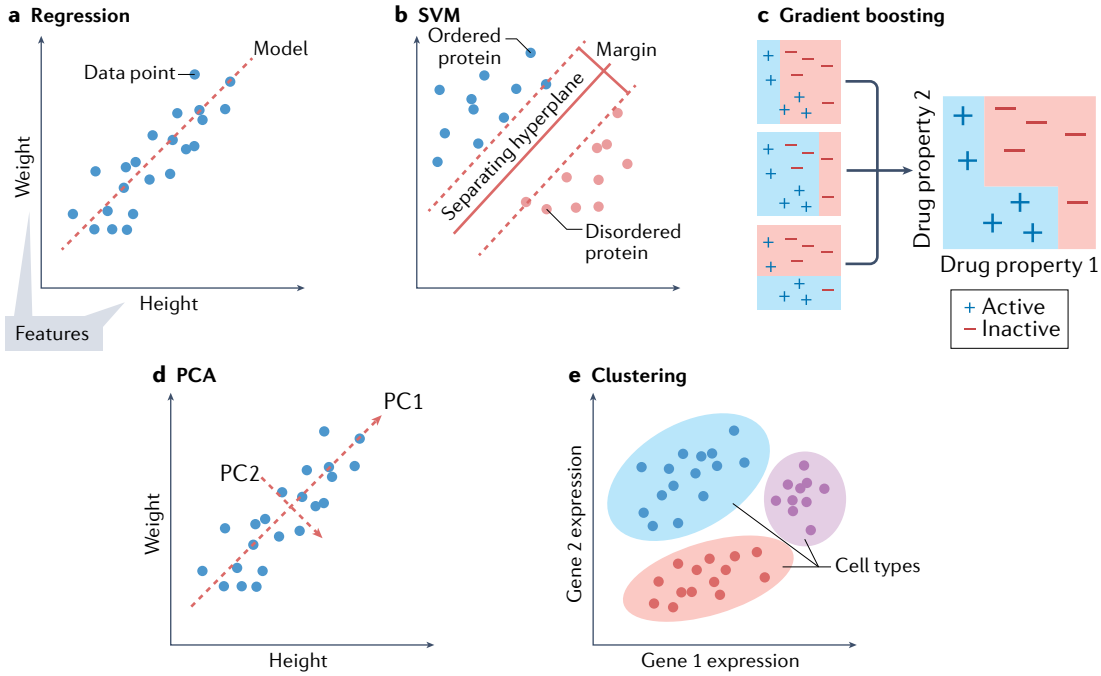

Fig. 3 | **Traditional machine learning methods. a** | Regression finds the relationship between a dependent variable (the observed property) and one or more independent variables (features); for example, predicting the weight of a person from the person's height. **b** | A support vector machine (SVM) transforms the original input data such that in their transformed versions (called the 'latent representation') data belonging to separate categories are divided by a clear gap that is made as wide as possible. In this case we show a prediction of whether a protein is ordered or disordered, with the axes representing dimensions of the transformed data. **c** | Gradient boosting uses an ensemble of weak prediction models, typically decision trees, to make predictions. For example, active drugs can be predicted from molecular descriptors such as molecular weight and the presence of particular chemical groups. Individual predictors are combined in a stage-wise manner to make the final prediction. **d** | Principal component analysis (PCA) finds a series of feature combinations that best describe the data while being orthogonal to each other. It is commonly used for dimensionality reduction. In the case of the height and weight of a person, the first principal component (PC1), corresponding to a linear combination of height and weight, describes the strong positive correlation, whereas PC2 might describe other variables that do not correlate strongly with those, such as percentage body fat or muscle mass. **e** | Clustering uses one of various algorithms to group sets of similar objects (for example, grouping cell types on the basis of gene expression profiles).

arrangement are often called 'multilayer perceptrons' and were the first networks useful for bioinformatics applications[52,53]. They are still widely used in a number of biological modelling applications today due to their ease and speed of training[13,54]. In many other applications, however, these simple architectures have been surpassed by newer model architectures discussed below, although some of these newer architectures still often make use of fully connected layers as subcomponents.

Convolutional neural networks. CNNs are ideally suited for image-like data, where the data possess some type of local structure, and where the recognition of such structure is a key objective of the analysis. With the example of images, this local structure could relate to specific types of objects in a field of view (for example, cells in a microscopy image), represented by specific local patterns of colours and/or edges in spatially close pixels in an input image.

CNNs are composed of one or more convolutional layers (see FIG. 4b), in which the output is the result of applying a small, one-layer fully connected neural network, called a 'filter' or 'kernel', to local groups of features in the input. In the case of image-like inputs, this local area would be a small patch of pixels in the image. The outputs of a convolutional layer are also image-like arrays, carrying the result of 'sliding' the filter over the entire input and computing an output at each position. Crucially, the same filter is used across all pixels, allowing the filters to learn local structure in the input data. It is common in deeper CNNs to use skip connections that allow the input signal to bypass one or more layers in addition to passing through the processing units in the layer. This type of network is called a 'residual network' and allows training to converge more quickly on accurate solutions.

CNNs can be configured to operate effectively on data of different spatial structure. For example, a 1D CNN would have filters that slide in just one direction (say from left to right); this type of CNN would be suitable for data that have only one spatial dimension (such as text or biological sequences). 2D CNNs operate on data with two spatial dimensions, such as digital images. 3D CNNs operate on volumetric data, such as magnetic resonance imaging scans.

CNNs have seen significant success in biology for a variety of data types. Recent advances in protein structure prediction have used information on the

co-evolution of residue pairs in related protein sequences to extract information on residue pair contacts and distances, allowing predictions of 3D protein structures to be built at unprecedented accuracy[23,55]. In this case, the network learns to pick out direct coupling interactions, and accurate predictions can be made even for sequences with few or no related sequences[56]. CNNs have also been applied successfully to identify variants in genetic sequence data[57], 3D genome folding[58], DNA–protein interactions[22,59], cryogenic electron microscopy image analysis[60,61] and image classification in medically important contexts (such as detection of malignancy), where they often now rival expert human performance[24,62].

Recurrent neural networks. RNNs are most suited to data that are in the form of ordered sequences, such that there exists (at least notionally) some dependence or correlation between one point in the sequence and the next. Probably their main application outside biology is in natural language processing, where text is treated as a sequence of words or characters. As shown in FIG. 4c, RNNs can be thought of as a block of neural network layers that take as input the data corresponding to each entry (or time step) in a sequence and produce an output for each entry that is dependent on entries that have previously been processed. They can also be used to generate a representation of the whole sequence that is passed to later layers of the network to generate the output. This is useful as sequences of any length can be converted to a fixed-size representation and input to a multilayer perceptron. Obvious examples for the use of RNNs in biology include analysis of gene or protein sequences, with tasks including identifying promoter regions from gene sequences, predicting protein secondary structure or modelling gene expression levels over time; in the last case, the value at a given time point would count as one entry in a sequence. The more advanced long short-term memory or gated recurrent unit variants of RNNs have many uses in biology, including protein structure prediction[63,64], peptide design[65] and predicting clinical diagnosis from health records[66]. These more advanced methods are often used in combination with CNNs,

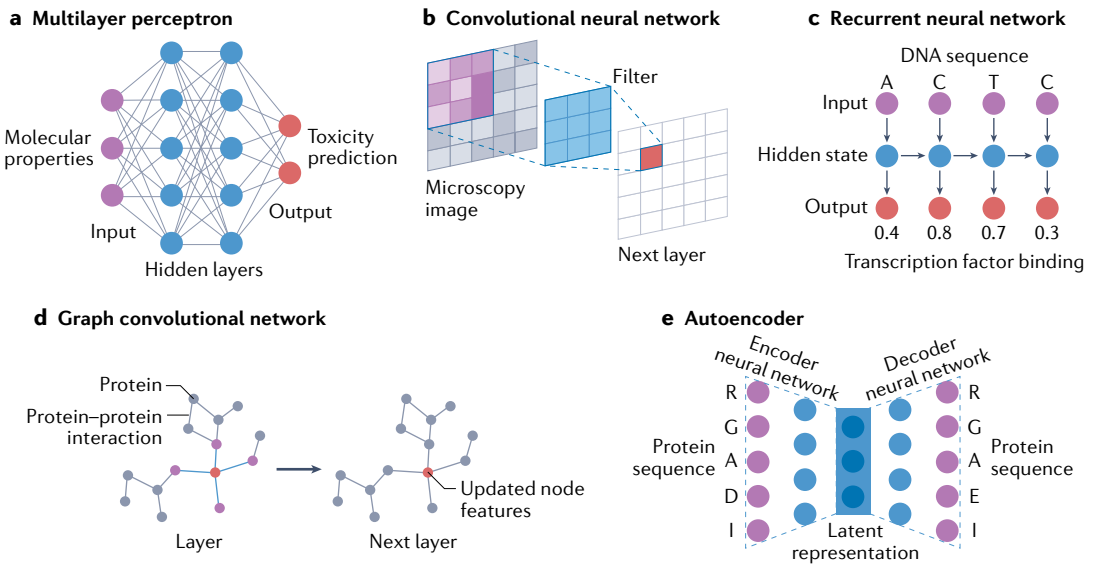

Fig. 4 | **Neural network methods. a** | A multilayer perceptron consists of nodes (shown as circles) that represent numbers: an input value, an output value or an internal (hidden) value. Nodes are arranged in layers with connections, indicating learned parameters, between every node of a layer and every node of the next layer. For example, molecular properties can be used to predict drug toxicity as the prediction can be made from some complicated combination of independent input features. **b** | A convolutional neural network (CNN) uses filters that move across the input layer and are used to calculate the values in the next layer. The filters operating across the whole layer mean that parameters are shared, allowing similar entities to be detected regardless of location. A 2D CNN is shown operating on a microscopy image, but 1D and 3D CNNs also find applications in biology. The dimensionality in this case refers to how many spatial dimensions there are in the data, and the connectivity within the CNNs can be configured accordingly. For example, biological sequences can be considered 1D and magnetic resonance imaging data can be considered 3D. **c** | A recurrent neural network (RNN) processes each part of a sequential input using the same learned parameters, giving an output and an updated hidden state for every input. The hidden state is used to carry information about the preceding parts of the sequence. In this case the probability of transcription factor binding is predicted for each base in a DNA sequence. The RNN is expanded to show how each output is generated using the same layers; this should not be confused with using different layers for each output. **d** | A graph convolutional network uses information from connected nodes in a graph, such as a protein–protein interaction network, to update node properties in the network by combining predictions from all neighbouring nodes. The updated node properties form the next layer in the network and predict the desired property in the output layer. **e** | An autoencoder consists of an encoder neural network, which converts an input into a lower-dimensional latent representation, and a decoder neural network, which converts this latent representation back to the original input form. For example, protein sequences can be encoded and the latent representation used to generate novel protein sequences. In the example, four of the five residues are the same as the input after encoding and decoding by the autoencoder, indicating an accuracy of 80% on this sequence.

which can increase accuracy[67]. RNNs can be very robust in analysing sequence-based data. For example, RNNs trained on millions of protein sequences have shown an ability to capture evolutionary and structural information, and can be applied to a variety of supervised tasks, including tasks related to the design of novel protein sequences[68].

Role of attention mechanisms and the use of transformers. A problem identified with RNNs is the difficulty they have in examining specific parts of an input sequence, which is important in order to generate a highly accurate output. The addition of an attention mechanism to RNNs, which allows the model to access all parts of the input sequence when calculating each output, was introduced to alleviate this problem. Recently it was shown that the RNN is not even required at all, and that attention alone can be used by itself; the resulting models, called 'transformers', have obtained state-of-the-art results on many natural language processing benchmarks[69]. Transformer models have recently shown greater accuracy than RNNs for tasks on biological sequences, but it remains to be seen whether these methods, which are often trained on billions of sequences using thousands of graphics processing units, will be able to outperform existing alignment-based methods of sequence analysis in bioinformatics[70]. The outstanding success of AlphaFold2 in the 14th Critical Assessment of Protein Structure Prediction (CASP14) experiment, a blind assessment of computational approaches to predict protein structure from sequence, suggests that models using attention also hold promise for tasks in structural biology[71].

Graph convolutional networks. Graph convolutional networks are particularly suitable for data that, while not having any obvious visible structure like an image, are nonetheless composed of entities connected by arbitrary specified relationships, or interactions[72]. Examples of such data relevant to biology include molecules (composed of atoms and bonds)[73–76] and protein–protein interaction networks (composed of proteins and interactions)[77]. A graph, in computational terms, is just a representation of such data, with each graph having a set of vertices or nodes, and a set of edges that represent various types of relationships or connections between the nodes. With use of the examples given above, representations of atoms or proteins might be classed as node features, and bonds or interactions might be classed as edge features. Graph convolutional networks use the structure of the resulting graph to determine the flow of information in the neural network model. As shown in FIG. 4d, adjacent nodes are considered when the features of each node are updated throughout the network, with the node features in the last layer being used as the output (for example, interacting residues on a protein) or combined to form an output for the whole graph (for example, fold type of the protein). Graphs representing different associations can combine different sources of information when making predictions, such as combining drug–gene and food–gene relationship graphs to predict foods for cancer prevention[78]. Software for training graph convolutional networks includes PyTorch Geometric[79] and Graph Nets[72].

Autoencoders. As the name suggests, autoencoders are neural network architectures designed to self-encode (autoencode) a collection of data points by representing them as points in a new space of predetermined dimensionality, usually far fewer than the number of input dimensions. One neural network (the encoder) is trained to convert the input into a compact internal representation, called a 'latent vector' or 'latent representation', representing a single point in the new space. The second part of an autoencoder, called the 'decoder', takes the latent vector as input and is trained to produce as output the original data with the original dimensionality (FIG. 4e). Another way of looking at this is that the encoder tries to compress the input, and the decoder tries to decompress it. The encoder, latent representation and decoder are trained at the same time. Although this sounds like a pointless exercise, where the output just mimics the input, the idea is to learn a new representation of the input data that compactly encodes desirable features, such as similarity between the data points, while still retaining the ability to reconstruct the original data using the learned latent representation. Applications include predicting how closely related two data points are and enforcing some structure on the latent space that is useful for further prediction tasks. Another benefit of the encoder–decoder architecture is that, once trained, the decoder can be used in isolation to generate predictions of new, synthetic data samples which can be tested in the laboratory and contribute to synthetic biology efforts[80]. Autoencoders have been applied to a range of biological problems, including predicting DNA methylation state[81], the engineering of gene and protein sequences[82,83] and single-cell RNA-sequencing analysis[84].

Training and improving neural networks. The general procedure for training machine learning models is outlined in BOX 1. However, as neural networks are structurally much more complex than the traditional machine learning algorithms, there are some concerns that are specific to neural networks. Having picked a neural network as an appropriate model for the intended application (FIG. 1), it is often a good idea to train it on just a single training example (for example, a single image or gene sequence). This trained model is not useful for making predictions, but the training is good at revealing programming errors. The training loss function should very quickly go to zero as the network simply memorizes the input; if it does not, there is likely an error in the code, or the algorithm is not complex enough to model the input data. Once the network has passed this basic debugging test, training on the whole training set can proceed, where the training loss function is minimized. This may require tuning of hyperparameters such as the learning rate (FIG. 2e). By monitoring loss on the training and validation sets, overfitting of the network can be detected where the training loss continues to drop lower and the loss on the validation set starts to increase. Training is usually stopped at that point, a process known as early stopping (FIG. 2f). Overfitting of a neural

Regularization
Restricting the values of parameters to prevent the model from overfitting to the training data. For example, penalizing high parameter values in regression models reduces the flexibility of the model and can stop it fitting to noise in the training data.

Cloud computing
On-demand computing services, including processing power and data storage, typically available via the Internet. A pay-as-you-go model is usually used. Use of cloud computing minimizes up-front IT infrastructure costs.

Hidden Markov model
A statistical model that can be used to describe the evolution of observable events that depend on factors that are not directly observable. It has various uses in biology, including representing protein sequence families.

network (or any machine learning model), visualized in FIG. 2d, means that the model is starting to simply memorize features of the training set and thus starting to lose its ability to generalize to new data. Early stopping is a good way of preventing this, but other techniques can be used during training, such as regularization of the model or dropout techniques, where nodes in the network are randomly ignored to force the network to learn a more robust prediction strategy involving multiple nodes.

Popular software packages used to train neural networks include PyTorch[85] and Tensorflow[86]. Training neural networks is computationally demanding, usually requiring a graphics processing unit or tensor processing unit with sufficient memory, as these devices can provide a 10 to 100 times speedup over use of the standard central processing unit. This speedup is required when training the larger models that have shown success in recent years, and when training is performed on large datasets. However, running an already trained model is usually considerably faster and is often feasible on just an ordinary central processing unit. Cloud computing solutions from common providers exist for those without access to a graphics processing unit for training, and it is worth noting that for small tasks, Colaboratory (Colab) allows Python code to be tested on either graphics processing units or tensor processing units free of charge. Using Colab is an excellent way of getting started with Python-based deep learning.

Challenges for biological applications

Perhaps the single biggest challenge of modelling biological data is the sheer variety[1]. Biologists work with data such as gene and protein sequences, gene expression levels over time, evolutionary trees, microscopy images, 3D structures and interaction networks, to name but a few. We have summarized some best practices and important considerations for specific biological data types in TABLE 2. Owing to the diversity of data types encountered, biological data often require somewhat bespoke solutions for handling them effectively, and this makes it difficult to recommend off-the-shelf tools or even general guidelines for the use of machine learning in these problem domains, as the choice of model, training procedure and test data will depend heavily on the exact questions one wants to answer. Nevertheless, there are some common issues that need to be considered for the successful use of machine learning in biology, but also more generally.

Data availability. Biology is somewhat unique in that there exist some problem domains that have very large quantities of data publicly available, whereas other problem domains have very small quantities. An example is the relative abundance of biological sequence data in public databases such as GenBank and UniProt, whereas reliable data on protein interactions are much harder to come by. The quantity of data available for a given problem has a profound impact on the choice of techniques that can effectively be used. As a very rough guideline, when only small amounts of data (hundreds of or a few thousand examples) are available, one is essentially forced to use more traditional machine learning methods, as these are more likely to produce robust predictions. When larger quantities are available, one can start to consider more highly parameterized models such as deep neural networks. In supervised machine learning, the relative proportions of each ground truth label in the dataset should also be considered, with more data required for machine learning to work if some labels are rare[87].

Data leakage. Although the scale and complexity of biological data may make them seem ideal for machine learning, there are some important considerations that need to be borne in mind[21,88,89]. One key concern is how to validate the performance of a model. The common setup of training, validation and test sets can lead to problems such as researchers repeatedly testing on the same test set with a variety of models to obtain the greatest accuracy, and hence risking overestimating performance on it without generalizing to other test sets or new data. However, biological data present a further non-trivial question: in a large dataset with related entries (for example, as a result of familial relationships, or evolutionary relationships), how does one ensure that two closely related entries do not end up split between the training set and the test set? If this occurs, then the ability of the model to remember specific cases is tested, rather than its ability to predict the property in question. This is one example of an issue often called 'data leakage' and leads to results that appear better than they are, which is perhaps one reason researchers are reluctant to be rigorous about the issue. Other types of data leakage are possible (for example, using any data or features during training that would not be available during testing). Here we focus on the problem of having related samples in the training and testing sets.

What we mean by 'related' here depends on the nature of the study. It might be a case of sampling data from the same patient or the same organism. However, the classic situation where data leakage occurs in biology is seen in studies on protein sequences and structures. Typically, but usually not correctly, researchers try to ensure that no protein in the training set has sequence identity above a certain threshold to any protein in the test set, usually at a threshold of 30% or 25%. This is enough to exclude many homologous pairs of proteins, but it has been known for decades that some homologous proteins can have virtually no sequence similarity[90,91], which would mean that simply filtering by sequence identity would be insufficient to prevent data leakage. This is particularly important for models that operate on sequence alignments or sequence profiles as input, as although two individual protein sequences may not share any obvious similarity, their profiles could be virtually identical. This means that for a machine learning model, these two profiles would essentially be the same data point — both will be describing the same protein family. For protein sequences, one solution to avoid this problem is to search the test data with a sensitive hidden Markov model profile comparison tool such as HH-suite, which can find sequences distantly related to the training data[92]. In the common case where protein structure is being used as input or output, structural classifications such as CATH[93] or ECOD[94] can be used to exclude similar folds

Table 2 | **Recommendations for the use of machine learning strategies for different biological data types**

Input data	Example prediction tasks	Recommended models	Challenges
Gene sequence	DNA accessibility[14] 3D genome organization[58] Enhancer–promoter interactions[40]	1D CNNs RNNs Transformers	Repetitive regions in genome Sparse regions of interest Very long sequences
Protein sequence	Protein structure[23,55] Protein function[132] Protein–protein interaction[133]	2D CNNs and residual networks using co-variation data Multilayer perceptrons with windowing Transformers	Metagenome data stored in many places and therefore hard to access Data leakage (from homology) can make validation difficult
Protein 3D structure	Protein model refinement[134] Protein model quality assessment[135] Change in stability upon mutation[136]	GCNs using molecular graph 3D CNNs using coordinates Traditional methods using structural features Clustering	Lack of data, particularly on protein complexes Lack of data on disordered proteins
Gene expression	Intergenic interactions or co-expression[137] Organization of transcription machinery[138]	Clustering CNNs Autoencoders	Unclear link between co-expression and function High dimensionality High noise
Mass spectrometry	Detecting peaks in spectra[139] Metabolite annotation[140]	CNNs using spectral data Traditional methods using derived features	Lack of standardized benchmarks[141] Normalization[a] required between different datasets
Images	Medical image recognition[24,62] Cryo-EM image reconstruction[60,142] RNA-sequencing profiles[143]	2D CNNs and residual networks Autoencoders Traditional methods using image features	Systematic differences in data collection affect prediction Hard to obtain large datasets of consistent data
Molecular structure	Antibiotic activity[73] Drug toxicity[54] Protein-ligand docking[39] Novel drug generation[144]	GCNs using molecular graph Traditional methods or multilayer perceptrons using molecular properties RNNs using text-based representations of molecular structure such as SMILES Autoencoders	Experimental data available for only a tiny fraction of possible small molecules
Protein–protein interaction network	Polypharmacology side effects[77] Protein function[145]	GCNs Graph embedding	Interaction networks can be incomplete Cellular location affects whether proteins interact High number of possible combinations

Each type of biological data has prediction tasks in which it has been used effectively, machine learning models that are appropriate and specific challenges when using machine learning. Some challenges, such as high dimensionality, affect most biological data types. CNN, convolutional neural network; cryo-EM, cryogenic electron microscopy; GCN, graph convolutional network; RNN, recurrent neural network. [a]'Normalization' means rescaling or otherwise transforming variables from different datasets with the intention that their contributions should carry roughly equal weight and their ranges are comparable on a joint scale. The most common way of achieving this is by subtracting the means of each variable and dividing by their standard deviations, which can also be called 'standardization'. This is required because different instruments, experimental protocols and so on can produce systematic differences in measurements of the same quantities, rendering it difficult or impossible to compare trends between experiments.

or homologous proteins. Similar issues affect studies predicting protein–ligand binding affinity[95].

To be clear, data leakage is not an intrinsic issue with any particular type of data, but rather it is a problem with how the data are used when training and evaluating machine learning models. One would certainly expect a trained model to produce very good results on data that are similar to the training set. The issue of data leakage becomes a problem when a model that appears accurate on some benchmark set performs poorly on new data that are actually different from the training set; in other words, the model does not generalize, likely because it has not modelled the true relationship between the variables, but rather remembered hidden associations present in the data.

Because of frequent complaints from reviewers, some journals are now starting to require rigorous benchmarking to be performed before a paper can be considered for publication. Without proper testing, the performance of a model will very likely not be representative of real-world performance on unseen data, which undermines user confidence in the model. Worse, authors of future studies may be misled into thinking that inadequate testing is defensible simply because it has already appeared in (possibly several) peer-reviewed articles, even though it is not. As mentioned in BOX 2, authors, peer reviewers and journal editors all have a responsibility for ensuring that data leakage has been avoided. Knowingly leaving these kinds of errors in place is really little better than fabricating data at the end of the day.

Interpretability of models. It is usually the case that biologists want to know why a particular model is making a particular prediction (that is, what features of the input

Saliency map
In the context of machine learning, an image generated to show which pixels in an input image contribute to the prediction made by a model. It is useful in interpreting models.

data the model is responding to and how) and why it works in some cases but not others. In other words, biologists are often interested in discovering mechanisms and the factors responsible for modelling output, rather than just accurate modelling, as mentioned previously. The ability to interpret a model depends on the machine learning method used and the input data. Interpretation is usually easier for non-neural network methods, as these have feature sets more amenable to direct meaningful interpretation and generally have fewer learnable parameters. In the case of, say, a simple linear regression model, the parameter assigned to each input feature gives a direct indication of how that feature affects the prediction.

Box 2 | **Evaluating articles that use machine learning**

Here are some questions to consider when reading or reviewing articles that use machine learning on biological data. It is useful to bear these considerations in mind, even if the answers are not apparent, and these questions can be used as the basis for a discussion with collaborators with the required expertise. A surprising number of articles do not fulfil these criteria[148].

Is the dataset adequately described?
Complete steps to assemble the dataset should be provided, ideally with the dataset or summary data (for example, biological database IDs) available at a persistent URL. In our experience, a thorough description of the machine learning method but with only a cursory reference to the data is a red flag. If a standard dataset or a dataset from another study is being used, then this should be adequately justified in the article.

Is the test set valid?
Based on the discussion in the section Challenges for biological applications, check that the test set is sufficient to benchmark the property under investigation. There should be no data leakage between the training set and the test set, the test set should be of large enough to give reliable results and the test set should mirror the range of examples a standard user of the tool would be likely to use it on. The composition and size of the training and test sets should be discussed in detail. Authors have a responsibility to ensure that all steps have been taken to avoid data leakage, and these steps should be described in the article, along with the rationale behind them. Journal editors and peer reviewers should also ensure that these tasks have been performed to a good standard, and certainly should never just assume that they have been.

Is the model choice justified?
Reasons should be given for the choice of machine learning method. Neural networks should be used because they are appropriate for the data and question in hand, and not just because everyone else is using them. Discussion of models that were tried and did not work should be encouraged as it may help others; too often a complex model is presented without any discussion of the inevitable trial and error that will have been required to end up with that model.

Has the method been compared with other methods?
A novel method should be compared with existing methods that show good performance and are used in the community. Ideally methods using a variety of model types should be compared, which can aid in interpreting results. It is surprising how many complex models can be matched in performance by simple regression methods.

Are the results too good to be true?
Claims of greater than 99% accuracy are not uncommon in machine learning articles in biology. Usually, this is a sign of a problem with the testing rather than an amazing breakthrough. Both authors and reviewers should take note of this point.

Is the method available?
At the very least, someone who wants to use a trained model from an article should be able to run a prediction using a Web service or binary file. Ideally, at least source code and the trained model should be available at a persistent URL and under a common licence[149,150]. Also making the training code available is the ideal scenario, as this further increases the reproducibility of the article and allows other researchers to build on the method without essentially having to start from scratch. Journals should bear some responsibility here to ensure that this becomes the norm.

The low cost of training non-neural network methods means that it is advisable to perform ablation studies, where the effect on performance of removing defined features of the input is measured. Ablation studies can reveal which features are most useful for the modelling task at hand, and are one way to possibly discover more robust, efficient and interpretable models.

Interpreting a neural network (particularly a deep neural network) is generally much harder due to the frequently large number of input features and parameters in the model. It is still possible to identify, for example, regions in an input image most responsible for a particular classification by building a saliency map[28]. Although saliency maps show which regions of an image are important, it can be more difficult to pinpoint which properties of the data in these locations were responsible for the prediction, particularly when the inputs are not easily interpretable by humans, such as images and text. Nevertheless, saliency maps and similar representations can be useful as a 'sanity check' to ensure that the model is indeed looking at the relevant parts of an image. This can help avoid situations where models make unintended connections, such as classifying medical images on the basis of hospital or department labels in the corner of the image rather than the medical content of the image itself[96]. Generating adversarial examples, synthetic inputs that cause a neural network to produce confident incorrect predictions, can also be a good way of providing information on which features are being most used for prediction[97]. For example, CNNs often use textures (such as stripes in animal fur) to classify objects in images, where humans would primarily use the shapes[51].

Privacy-preserving machine learning. Some biological data, most notably human genomics data and commercially sensitive pharmaceutical data, have data privacy implications. There have been a number of efforts to allow sharing of data and distributed training of machine learning models in the context of data privacy. For example, modern cryptographic techniques allow training of a drug–target interaction model where the data and results are provably secure[98]. Simulated, synthetic participants that closely resemble real participants in a clinical trial can lead to results that are accurate for real participants without revealing identifying data[99]. Algorithms have been developed for efficient federated model training with data stored in different places[100].

The need for interdisciplinary collaborations. Unless publicly available data are being used, it is rare that one research group will have the expertise and resources to both collect data for a machine learning study and also apply the most appropriate machine learning method effectively. It is common for experimental biologists to collaborate with computer scientists, with such collaborations often obtaining excellent results. It is, however, important in such collaborations that each side has some working knowledge of the other. In particular, computer scientists should make an effort to understand the data, such as the expected degree of noise and reproducibility, and biologists should understand the limitations of the machine learning algorithms being used. Building such

Automatic differentiation
A set of techniques to automatically calculate the gradient of a function in a computer program. Used to train neural networks, where it is called 'backpropagation'.

Gradients
The rate of change of one property as another property changes. In neural networks, the set of gradients of the loss function with respect to the neural network parameters, computed via a process known as backpropagation, is used to adjust the parameters and thus train the model.

understanding takes time and effort, but is important to prevent the often unintentional dissemination of poor models and misleading results.

Future directions

The increasing use of machine learning in biological studies looks set to continue for the foreseeable future. This increased uptake has been enabled by important advances in methodology, software and hardware, all of which keep on developing. A number of large technology companies are using their technical expertise and considerable resources to assist academic researchers or even perform their own research in biology with innovative machine learning strategies. To date, however, most success has come from applying algorithms developed in other fields directly to biological data. For example, CNNs and RNNs were developed for applications such as image analysis (for face recognition or in self-driving cars) and natural language processing, respectively. One of the most exciting prospects for machine learning in biology is algorithms tailored specifically to biological data and biological questions[101,102]. If the known structure of a biological system can be exploited and neural networks used to learn the unknown parts, then increasingly heavily parameterized models can be replaced with simpler ones that are more amenable to interpretation and more robust on new data[103]. Applications include biological reaction systems and pharmacokinetics, where systems of known differential equations can be used. This will also assist in the move from predictive machine learning to generative models that can create new entities, such as designing proteins with novel structures and functions[104,105].

As the variety of useful architectures and input data types increases, the paradigm of differentiable programming is emerging from the field of deep learning[106]. Differentiable programming is the use of automatic differentiation, the central concept in training neural networks, to calculate gradients and improve parameters in any desired algorithm. This shows promise for physical models of biological systems in protein structure prediction[63,107], and for learning force field parameters for molecular dynamics simulations[108,109]. The development of differentiable software packages such as JAX[110] and packages tailored to specific areas of biology such as Selene[111], Janggu[112] and JAX MD[113] will assist the development of such methods.

The progress in biological data analysis with machine learning has also been enabled by the deposition of trained models in publicly available repositories. In this way, researchers working on similar problems can use these models without the need for training, and a variety of different models can be used with only minimal effort required for switching between them[114]. The field has also seen an expansion of automated machine learning pipelines, which train and tune a variety of models without user input and return the best performing model. These may assist non-experts in training models[115]. However, these resources cannot replace a thorough understanding of the method being used, which is important for choosing the appropriate inductive bias and interpreting the predictions of the model. It remains to be seen whether in the future automated machine learning will be reliable and flexible enough to allow experimentalists to routinely use complex machine learning algorithms independently, or whether machine learning expertise will remain a necessity.

As has been discussed, rigorous validation of models and comparison of different models is challenging but remains necessary to identify the best performing models and inform future research directions. For the field to progress, it will be necessary to develop benchmark datasets and validation tasks[116], such as ProteinNet[117], ATOM3D[118] and TAPE[119], and for these to become widely used. Of course, overoptimizing to a particular benchmark can occur, and it is important that researchers resist the temptation to do this to make their results seem better. Blind assessments such as CASP[120] and the Critical Assessment of Functional Annotation[121] will continue to play an important role in assessing which models perform best.

Overall, the variety of biological data makes it hard to provide general guidelines for machine learning in biology. Hence, we have aimed here to give biologists an overview of the different methods available and to provide them with some ideas about how to conduct effective machine learning with their data. Of course, machine learning is not suited to every problem, and it is just as important to know when to avoid it: when there are not sufficient data, when understanding rather than prediction is required or when it is unclear how to assess performance in a fair way. The boundaries of when machine learning is useful in biology are still being explored and will continue to change in accordance with the nature and volume of available experimental data. Undeniably, though, machine learning has had a huge impact on biology and will continue to do so.

Published online 13 September 2021

1. Ching, T. et al. Opportunities and obstacles for deep learning in biology and medicine. *J. R. Soc. Interface* **15**, 20170387 (2018).
This is a thorough review of applications of deep learning to biology and medicine including many references to the literature.
2. Mitchell, T. M. Machine Learning (McGraw Hill, 1997).
3. Goodfellow, I., Bengio Y. & Courville, A. Deep Learning (MIT Press, 2016).
4. Libbrecht, M. W. & Noble, W. S. Machine learning applications in genetics and genomics. *Nat. Rev. Genet.* **16**, 321–332 (2015).
5. Zou, J. et al. A primer on deep learning in genomics. *Nat. Genet.* **51**, 12–18 (2019).
6. Myszczynska, M. A. et al. Applications of machine learning to diagnosis and treatment of neurodegenerative diseases. *Nat. Rev. Neurol.* **16**, 440–456 (2020).
7. Yang, K. K., Wu, Z. & Arnold, F. H. Machine-learning-guided directed evolution for protein engineering. *Nat. Methods* **16**, 687–694 (2019).
8. Tarca, A. L., Carey, V. J., Chen, X.-W., Romero, R. & Drăghici, S. Machine learning and its applications to biology. *PLoS Comput. Biol.* **3**, e116 (2007).
This is an introduction to machine learning concepts and applications in biology with a focus on traditional machine learning methods.
9. Silva, J. C. F., Teixeira, R. M., Silva, F. F., Brommonschenkel, S. H. & Fontes, E. P. B. Machine learning approaches and their current application in plant molecular biology: a systematic review. *Plant. Sci.* **284**, 37–47 (2019).
10. Kandoi, G., Acencio, M. L. & Lemke, N. Prediction of druggable proteins using machine learning and systems biology: a mini-review. *Front. Physiol.* **6**, 366 (2015).
11. Marblestone, A. H., Wayne, G. & Kording, K. P. Toward an integration of deep learning and neuroscience. *Front. Comput. Neurosci.* **10**, 94 (2016).
12. Jiménez-Luna, J., Grisoni, F. & Schneider, G. Drug discovery with explainable artificial intelligence. *Nat. Mach. Intell.* **2**, 573–584 (2020).
13. Buchan, D. W. A. & Jones, D. T. The PSIPRED Protein Analysis Workbench: 20 years on. *Nucleic Acids Res.* **47**, W402–W407 (2019).
14. Kelley, D. R., Snoek, J. & Rinn, J. L. Basset: learning the regulatory code of the accessible genome with deep convolutional neural networks. *Genome Res.* **26**, 990–999 (2016).

15. Altman, N. & Krzywinski, M. Clustering. *Nat. Methods* **14**, 545–546 (2017).
16. Hopf, T. A. et al. Mutation effects predicted from sequence co-variation. *Nat. Biotechnol.* **35**, 128–135 (2017).
17. Zhang, Z. et al. Predicting folding free energy changes upon single point mutations. *Bioinformatics* **28**, 664–671 (2012).
18. Pedregosa, F. et al. Scikit-learn: machine learning in python. *J. Mach. Learn. Res.* **12**, 2825–2830 (2011).
19. Kuhn, M. Building predictive models in r using the caret package. *J. Stat. Softw.* **28**, 1–26 (2008).
20. Blaom, A. D. et al. MLJ: a Julia package for composable machine learning. *J. Open Source Softw.* **5**, 2704 (2020).
21. Jones, D. T. Setting the standards for machine learning in biology. *Nat. Rev. Mol. Cell Biol.* **20**, 659–660 (2019).
22. Alipanahi, B., Delong, A., Weirauch, M. T. & Frey, B. J. Predicting the sequence specificities of DNA- and RNA-binding proteins by deep learning. *Nat. Biotechnol.* **33**, 831–838 (2015).
23. Senior, A. W. et al. Improved protein structure prediction using potentials from deep learning. *Nature* **577**, 706–710 (2020).
Technology company DeepMind entered the CASP13 assessment in protein structure prediction and its method using deep learning was the most accurate of the methods entered.
24. Esteva, A. et al. Dermatologist-level classification of skin cancer with deep neural networks. *Nature* **542**, 115–118 (2017).
25. Tegunov, D. & Cramer, P. Real-time cryo-electron microscopy data preprocessing with Warp. *Nat. Methods* **16**, 1146–1152 (2019).
26. LeCun, Y., Bengio, Y. & Hinton, G. Deep learning. *Nature* **521**, 436–444 (2015).
This is a review of deep learning by some of the major figures in the deep learning revolution.
27. Hastie T., Tibshirani R., Friedman J. The elements of statistical learning: data mining, inference, and prediction. 2nd Edn. (Springer Science & Business Media; 2009).
28. Adebayo, J. et al. Sanity checks for saliency maps. *NeurIPS* https://arxiv.org/abs/1810.03292 (2018).
29. Gal, Y. & Ghahramani, Z. Dropout as a Bayesian approximation: representing model uncertainty in deep learning. *ICML* **48**, 1050–1059 (2016).
30. Smith, A. M. et al. Standard machine learning approaches outperform deep representation learning on phenotype prediction from transcriptomics data. *BMC Bioinformatics* **21**, 119 (2020).
31. Tibshirani, R. Regression shrinkage and selection via the lasso. *J. R. Stat. Soc. Ser. B.* **58**, 267–288 (1996).
32. Zou, H. & Hastie, T. Regularization and variable selection via the elastic net. *J. R. Stat. Soc. Ser. B.* **67**, 301–320 (2005).
33. Noble, W. S. What is a support vector machine? *Nat. Biotechnol.* **24**, 1565–1567 (2006).
34. Ben-Hur, A. & Weston, J. A user's guide to support vector machines. *Methods Mol. Biol.* **609**, 223–239 (2010).
35. Ben-Hur, A., Ong, C. S., Sonnenburg, S., Schölkopf, B. & Rätsch, G. Support vector machines and kernels for computational biology. *PLoS Comput. Biol.* **4**, e1000173 (2008).
This is an introduction to SVMs with a focus on biological data and prediction tasks.
36. Kircher, M. et al. A general framework for estimating the relative pathogenicity of human genetic variants. *Nat. Genet.* **46**, 310–315 (2014).
37. Driscoll, M. K. et al. Robust and automated detection of subcellular morphological motifs in 3D microscopy images. *Nat. Methods* **16**, 1037–1044 (2019).
38. Bzdok, D., Krzywinski, M. & Altman, N. Machine learning: supervised methods. *Nat. Methods* **15**, 5–6 (2018).
39. Wang, C. & Zhang, Y. Improving scoring-docking-screening powers of protein-ligand scoring functions using random forest. *J. Comput. Chem.* **38**, 169–177 (2017).
40. Zeng, W., Wu, M. & Jiang, R. Prediction of enhancer-promoter interactions via natural language processing. *BMC Genomics* **19**, 84 (2018).
41. Olson, R. S., Cava, W. L., Mustahsan, Z., Varik, A. & Moore, J. H. Data-driven advice for applying machine learning to bioinformatics problems. *Pac. Symp. Biocomput.* **23**, 192–203 (2018).
42. Rappoport, N. & Shamir, R. Multi-omic and multi-view clustering algorithms: review and cancer benchmark. *Nucleic Acids Res.* **47**, 1044 (2019).
43. Steinegger, M. & Söding, J. MMseqs2 enables sensitive protein sequence searching for the analysis of massive data sets. *Nat. Biotechnol.* **35**, 1026–1028 (2017).
44. Jain, A. K. Data clustering: 50 years beyond K-means. *Pattern Recognit. Lett.* **31**, 651–666 (2010).
45. Ester M., Kriegel H.-P., Sander J., Xu X. A density-based algorithm for discovering clusters in large spatial databases with noise. *KDD'96 Proc. Second Int. Conf. Knowl. Discov. Data Mining.* **96**, 226–231 (1996).
46. Nguyen, L. H. & Holmes, S. Ten quick tips for effective dimensionality reduction. *PLoS Comput. Biol.* **15**, e1006907 (2019).
47. Moon, K. R. et al. Visualizing structure and transitions in high-dimensional biological data. *Nat. Biotechnol.* **37**, 1482–1492 (2019).
48. van der Maaten, L. & Hinton, G. Visualizing data using t-SNE. *J. Mach. Learn. Res.* **9**, 2579–2605 (2008).
49. Kobak, D. & Berens, P. The art of using t-SNE for single-cell transcriptomics. *Nat. Commun.* **10**, 5416 (2019).
This article provides a discussion and tips for using *t*-SNE as a dimensionality reduction technique on single-cell transcriptomics data.
50. Crick, F. The recent excitement about neural networks. *Nature* **337**, 129–132 (1989).
51. Geirhos, R. et al. Shortcut learning in deep neural networks. *Nat. Mach. Intell.* **2**, 665–673 (2020).
This article discusses a common problem in deep learning called 'shortcut learning', where the model uses decision rules that do not transfer to real-world data.
52. Qian, N. & Sejnowski, T. J. Predicting the secondary structure of globular proteins using neural network models. *J. Mol. Biol.* **202**, 865–884 (1988).
53. deFigueiredo, R. J. et al. Neural-network-based classification of cognitively normal, demented, Alzheimer disease and vascular dementia from single photon emission with computed tomography image data from brain. *Proc. Natl Acad. Sci. USA* **92**, 5530–5534 (1995).
54. Mayr, A., Klambauer, G., Unterthiner, T. & Hochreiter, S. DeepTox: toxicity prediction using deep learning. *Front. Environ. Sci.* **3**, 80 (2016).
55. Yang, J. et al. Improved protein structure prediction using predicted interresidue orientations. *Proc. Natl Acad. Sci. USA* **117**, 1496–1503 (2020).
56. Xu, J., Mcpartlon, M. & Li, J. Improved protein structure prediction by deep learning irrespective of co-evolution information. *Nat. Mach. Intell.* **3**, 601–609 (2021).
57. Poplin, R. et al. A universal SNP and small-indel variant caller using deep neural networks. *Nat. Biotechnol.* **36**, 983–987 (2018).
58. Fudenberg, G., Kelley, D. R. & Pollard, K. S. Predicting 3D genome folding from DNA sequence with Akita. *Nat. Methods* **17**, 1111–1117 (2020).
59. Zeng, H., Edwards, M. D., Liu, G. & Gifford, D. K. Convolutional neural network architectures for predicting DNA-protein binding. *Bioinformatics* **32**, i121–i127 (2016).
60. Yao, R., Qian, J. & Huang, Q. Deep-learning with synthetic data enables automated picking of cryo-EM particle images of biological macromolecules. *Bioinformatics* **36**, 1252–1259 (2020).
61. Si, D. et al. Deep learning to predict protein backbone structure from high-resolution cryo-EM density maps. *Sci. Rep.* **10**, 4282 (2020).
62. Poplin, R. et al. Prediction of cardiovascular risk factors from retinal fundus photographs via deep learning. *Nat. Biomed. Eng.* **2**, 158–164 (2018).
63. AlQuraishi, M. End-to-end differentiable learning of protein structure. *Cell Syst.* **8**, 292–301.e3 (2019).
64. Heffernan, R., Yang, Y., Paliwal, K. & Zhou, Y. Capturing non-local interactions by long short-term memory bidirectional recurrent neural networks for improving prediction of protein secondary structure, backbone angles, contact numbers and solvent accessibility. *Bioinformatics* **33**, 2842–2849 (2017).
65. Müller, A. T., Hiss, J. A. & Schneider, G. Recurrent neural network model for constructive peptide design. *J. Chem. Inf. Model.* **58**, 472–479 (2018).
66. Choi, E., Bahadori, M. T., Schuetz, A., Stewart, W. F. & Sun, J. Doctor AI: predicting clinical events via recurrent neural networks. *JMLR Workshop Conf. Proc.* **56**, 301–318 (2016).
67. Quang, D. & Xie, X. DanQ: a hybrid convolutional and recurrent deep neural network for quantifying the function of DNA sequences. *Nucleic Acids Res.* **44**, e107 (2016).
68. Alley, E. C., Khimulya, G., Biswas, S., AlQuraishi, M. & Church, G. M. Unified rational protein engineering with sequence-based deep representation learning. *Nat. Methods* **16**, 1315–1322 (2019).
69. Vaswani, A. et al. Attention is all you need. *arXiv* https://arxiv.org/abs/1706.03762 (2017).
70. Elnaggar, A. et al. ProtTrans: towards cracking the language of life's code through self-supervised deep learning and high performance computing. *arXiv* https://arxiv.org/abs/2007.06225 (2020).
71. Jumper, J. et al. Highly accurate protein structure prediction with AlphaFold. *Nature* **596**, 583–589 (2021).
72. Battaglia, P. W. et al. Relational inductive biases, deep learning, and graph networks. *arXiv* https://arxiv.org/abs/1806.01261 (2018).
73. Stokes, J. M. et al. A deep learning approach to antibiotic discovery. *Cell* **181**, 475–483 (2020).
In this work, a deep learning model predicts antibiotic activity, with one candidate showing broad-spectrum antibiotic activities in mice.
74. Gainza, P. et al. Deciphering interaction fingerprints from protein molecular surfaces using geometric deep learning. *Nat. Methods* **17**, 184–192 (2020).
75. Strokach, A., Becerra, D., Corbi-Verge, C., Perez-Riba, A. & Kim, P. M. Fast and flexible protein design using deep graph neural networks. *Cell Syst.* **11**, 402–411.e4 (2020).
76. Gligorijevic, V. et al. Structure-based function prediction using graph convolutional networks. *Nat. Commun.* **12**, 3168 (2021).
77. Zitnik, M., Agrawal, M. & Leskovec, J. Modeling polypharmacy side effects with graph convolutional networks. *Bioinformatics* **34**, i457–i466 (2018).
78. Veselkov, K. et al. HyperFoods: machine intelligent mapping of cancer-beating molecules in foods. *Sci. Rep.* **9**, 9237 (2019).
79. Fey, M. & Lenssen, J. E. Fast graph representation learning with PyTorch geometric. *arXiv* https://arxiv.org/abs/1903.02428 (2019).
80. Zhavoronkov, A. et al. Deep learning enables rapid identification of potent DDR1 kinase inhibitors. *Nat. Biotechnol.* **37**, 1038–1040 (2019).
81. Wang, Y. et al. Predicting DNA methylation state of CpG dinucleotide using genome topological features and deep networks. *Sci. Rep.* **6**, 19598 (2016).
82. Linder, J., Bogard, N., Rosenberg, A. B. & Seelig, G. A generative neural network for maximizing fitness and diversity of synthetic DNA and protein sequences. *Cell Syst.* **11**, 49–62.e16 (2020).
83. Greener, J. G., Moffat, L. & Jones, D. T. Design of metalloproteins and novel protein folds using variational autoencoders. *Sci. Rep.* **8**, 16189 (2018).
84. Wang, J. et al. scGNN is a novel graph neural network framework for single-cell RNA-Seq analyses. *Nat. Commun.* **12**, 1882 (2021).
85. Paszke, A. et al. PyTorch: an imperative style, high-performance deep learning library. *Adv. Neural Inf. Process. Syst.* **32**, 8024–8035 (2019).
86. Abadi M. et al. Tensorflow: a system for large-scale machine learning. 12th USENIX Symposium on Operating Systems Design and Implementation. 265–283 (USENIX, 2016).
87. Wei, Q. & Dunbrack, R. L. Jr The role of balanced training and testing data sets for binary classifiers in bioinformatics. *PLoS ONE* **8**, e67863 (2013).
88. Walsh, I., Pollastri, G. & Tosatto, S. C. E. Correct machine learning on protein sequences: a peer-reviewing perspective. *Brief. Bioinform* **17**, 831–840 (2016).
This article discusses how peer reviewers can assess machine learning methods in biology, and by extension how scientists can design and conduct such studies properly.
89. Schreiber, J., Singh, R., Bilmes, J. & Noble, W. S. A pitfall for machine learning methods aiming to predict across cell types. *Genome Biol.* **21**, 282 (2020).
90. Chothia, C. & Lesk, A. M. The relation between the divergence of sequence and structure in proteins. *EMBO J.* **5**, 823–826 (1986).
91. Söding, J. & Remmert, M. Protein sequence comparison and fold recognition: progress and good-practice benchmarking. *Curr. Opin. Struct. Biol.* **21**, 404–411 (2011).
92. Steinegger, M. et al. HH-suite3 for fast remote homology detection and deep protein annotation. *BMC Bioinformatics* **20**, 473 (2019).
93. Sillitoe, I. et al. CATH: expanding the horizons of structure-based functional annotations for genome sequences. *Nucleic Acids Res.* **47**, D280–D284 (2019).

94. Cheng, H. et al. ECOD: an evolutionary classification of protein domains. *PLoS Comput. Biol.* **10**, e1003926 (2014).
95. Li, Y. & Yang, J. Structural and sequence similarity makes a significant impact on machine-learning-based scoring functions for protein-ligand interactions. *J. Chem. Inf. Model.* **57**, 1007–1012 (2017).
96. Zech, J. R. et al. Variable generalization performance of a deep learning model to detect pneumonia in chest radiographs: a cross-sectional study. *PLoS Med.* **15**, e1002683 (2018).
97. Szegedy, C. et al. Intriguing properties of neural networks. *arXiv* https://arxiv.org/abs/1312.6199 (2014).
98. Hie, B., Cho, H. & Berger, B. Realizing private and practical pharmacological collaboration. *Science* **362**, 347–350 (2018).
99. Beaulieu-Jones, B. K. et al. Privacy-preserving generative deep neural networks support clinical data sharing. *Circ. Cardiovasc. Qual. Outcomes* **12**, e005122 (2019).
100. Konečný, J., Brendan McMahan, H., Ramage, D. & Richtárik, P. Federated optimization: distributed machine learning for on-device intelligence. *arXiv* https://arxiv.org/abs/1610.02527 (2016).
101. Pérez, A., Martínez-Rosell, G. & De Fabritiis, G. Simulations meet machine learning in structural biology. *Curr. Opin. Struct. Biol.* **49**, 139–144 (2018).
102. Noé, F., Olsson, S., Köhler, J. & Wu, H. Boltzmann generators: sampling equilibrium states of many-body systems with deep learning. *Science* **365**, 6457 (2019).
103. Shrikumar, A., Greenside, P. & Kundaje, A. Reverse-complement parameter sharing improves deep learning models for genomics. *bioRxiv* https://www.biorxiv.org/content/10.1101/103663v1 (2017).
104. Lopez, R., Gayoso, A. & Yosef, N. Enhancing scientific discoveries in molecular biology with deep generative models. *Mol. Syst. Biol.* **16**, e9198 (2020).
105. Anishchenko, I., Chidyausiku, T. M., Ovchinnikov, S., Pellock, S. J. & Baker, D. De novo protein design by deep network hallucination. *bioRxiv* https://doi.org/10.1101/2020.07.22.211482 (2020).
106. Innes, M. et al. A differentiable programming system to bridge machine learning and scientific computing. *arXiv* https://arxiv.org/abs/1907.07587 (2019).
107. Ingraham J., Riesselman A. J., Sander C., Marks D. S. Learning protein structure with a differentiable simulator. *ICLR* https://openreview.net/forum?id = Byg3y3C9Km (2019).
108. Jumper, J. M., Faruk, N. F., Freed, K. F. & Sosnick, T. R. Trajectory-based training enables protein simulations with accurate folding and Boltzmann ensembles in cpu-hours. *PLoS Comput. Biol.* **14**, e1006578 (2018).
109. Wang, Y., Fass, J. & Chodera, J. D. End-to-end differentiable molecular mechanics force field construction. *arXiv* http://arxiv.org/abs/2010.01196 (2020).
110. Bradbury, J. et al. JAX: composable transformations of Python + NumPy programs. *GitHub* http://github.com/google/jax (2018).
111. Chen, K. M., Cofer, E. M., Zhou, J. & Troyanskaya, O. G. Selene: a PyTorch-based deep learning library for sequence data. *Nat. Methods* **16**, 315–318 (2019).
This work provides a software library based on PyTorch providing functionality for biological sequences.
112. Kopp, W., Monti, R., Tamburrini, A., Ohler, U. & Akalin, A. Deep learning for genomics using Janggu. *Nat. Commun.* **11**, 3488 (2020).
113. Schoenholz, S. S. & Cubuk, E. D. JAX, M.D.: end-to-end differentiable, hardware accelerated, molecular dynamics in pure Python. *arXiv* https://arxiv.org/abs/1912.04232 (2019).
114. Avsec, Ž. et al. The Kipoi repository accelerates community exchange and reuse of predictive models for genomics. *Nat. Biotechnol.* **37**, 592–600 (2019).
115. Isensee, F., Jaeger, P. F., Kohl, S. A. A., Petersen, J. & Maier-Hein, K. H. nnU-Net: a self-configuring method for deep learning-based biomedical image segmentation. *Nat Methods* **18**, 203–211 (2020).
116. Livesey, B. J. & Marsh, J. A. Using deep mutational scanning to benchmark variant effect predictors and identify disease mutations. *Mol. Syst. Biol.* **16**, e9380 (2020).
117. AlQuraishi, M. ProteinNet: a standardized data set for machine learning of protein structure. *BMC Bioinformatics* **20**, 311 (2019).
118. Townshend, R. J. L. et al. ATOM3D: tasks on molecules in three dimensions. *arXiv* https://arxiv.org/abs/2012.04035 (2020).
119. Rao, R. et al. Evaluating protein transfer learning with TAPE. *Adv. Neural. Inf. Process. Syst.* **32**, 9689–9701 (2019).
120. Kryshtafovych, A., Schwede, T., Topf, M., Fidelis, K. & Moult, J. Critical assessment of methods of protein structure prediction (CASP) — round XIII. *Proteins* **87**, 1011–1020 (2019).
121. Zhou, N. et al. The CAFA challenge reports improved protein function prediction and new functional annotations for hundreds of genes through experimental screens. *Genome Biol.* **20**, 244 (2019).
122. Munro, D. & Singh, M. DeMaSk: a deep mutational scanning substitution matrix and its use for variant impact prediction. *Bioinformatics* **36**, 5322–5329 (2020).
123. Haario, H. & Taavitsainen, V.-M. Combining soft and hard modelling in chemical kinetic models. *Chemom. Intell. Lab. Syst.* **44**, 77–98 (1998).
124. Cozzetto, D., Minneci, F., Currant, H. & Jones, D. T. FFPred 3: feature-based function prediction for all gene ontology domains. *Sci. Rep.* **6**, 31865 (2016).
125. Nugent, T. & Jones, D. T. Transmembrane protein topology prediction using support vector machines. *BMC Bioinformatics* **10**, 159 (2009).
126. Bao, L., Zhou, M. & Cui, Y. nsSNPAnalyzer: identifying disease-associated nonsynonymous single nucleotide polymorphisms. *Nucleic Acids Res.* **33**, W480–W482 (2005).
127. Li, W., Yin, Y., Quan, X. & Zhang, H. Gene expression value prediction based on XGBoost algorithm. *Front. Genet.* **10**, 1077 (2019).
128. Zhang, Y. & Skolnick, J. SPICKER: a clustering approach to identify near-native protein folds. *J. Comput. Chem.* **30**, 865–871 (2004).
129. Teodoro, M. L., Phillips, G. N. Jr & Kavraki, L. E. Understanding protein flexibility through dimensionality reduction. *J. Comput. Biol.* **10**, 617–634 (2003).
130. Schlichtkrull, M. et al. Modeling relational data with graph convolutional networks. *arXiv* https://arxiv.org/abs/1703.06103 (2019).
131. Pandarinath, C. et al. Inferring single-trial neural population dynamics using sequential auto-encoders. *Nat. Methods* **15**, 805–815 (2018).
132. Antczak, M., Michaelis, M. & Wass, M. N. Environmental conditions shape the nature of a minimal bacterial genome. *Nat. Commun.* **10**, 3100 (2019).
133. Sun, T., Zhou, B., Lai, L. & Pei, J. Sequence-based prediction of protein protein interaction using a deep-learning algorithm. *BMC Bioinformatics* **18**, 277 (2017).
134. Hiranuma, N. et al. Improved protein structure refinement guided by deep learning based accuracy estimation. *Nat. Commun.* **12**, 1340 (2021).
135. Pagès, G., Charmettant, B. & Grudinin, S. Protein model quality assessment using 3D oriented convolutional neural networks. *Bioinformatics* **35**, 3313–3319 (2019).
136. Pires, D. E. V., Ascher, D. B. & Blundell, T. L. DUET: a server for predicting effects of mutations on protein stability using an integrated computational approach. *Nucleic Acids Res.* **42**, W314–W319 (2014).
137. Yuan, Y. & Bar-Joseph, Z. Deep learning for inferring gene relationships from single-cell expression data. *Proc. Natl Acad. Sci. USA* **116**, 27151–27158 (2019).
138. Chen, L., Cai, C., Chen, V. & Lu, X. Learning a hierarchical representation of the yeast transcriptomic machinery using an autoencoder model. *BMC Bioinformatics* **17**, S9 (2016).
139. Kantz, E. D., Tiwari, S., Watrous, J. D., Cheng, S. & Jain, M. Deep neural networks for classification of LC-MS spectral peaks. *Anal. Chem.* **91**, 12407–12413 (2019).
140. Dührkop, K. et al. SIRIUS 4: a rapid tool for turning tandem mass spectra into metabolite structure information. *Nat. Methods* **16**, 299–302 (2019).
141. Liebal, U. W., Phan, A. N. T., Sudhakar, M., Raman, K. & Blank, L. M. Machine learning applications for mass spectrometry-based metabolomics. *Metabolites* **10**, 243 (2020).
142. Zhong, E. D., Bepler, T., Berger, B. & Davis, J. H. CryoDRGN: reconstruction of heterogeneous cryo-EM structures using neural networks. *Nat. Methods* **18**, 176–185 (2021).
143. Schmauch, B. et al. A deep learning model to predict RNA-Seq expression of tumours from whole slide images. *Nat. Commun.* **11**, 3877 (2020).
144. Das, P. et al. Accelerated antimicrobial discovery via deep generative models and molecular dynamics simulations. *Nat. Biomed. Eng.* **5**, 613–623 (2021).
145. Gligorijevic, V., Barot, M. & Bonneau, R. deepNF: deep network fusion for protein function prediction. *Bioinformatics* **34**, 3873–3881 (2018).
146. Karpathy A. A recipe for training neural networks. https://karpathy.github.io/2019/04/25/recipe (2019).
147. Bengio, Y. Practical recommendations for gradient-based training of deep architectures. *Lecture Notes Comput. Sci.* **7700**, 437–478 (2012).
148. Roberts, M. et al. Common pitfalls and recommendations for using machine learning to detect and prognosticate for COVID-19 using chest radiographs and CT scans. *Nat. Mach. Intell.* **3**, 199–217 (2021).
This study assesses 62 machine learning studies that analyse medical images for COVID-19 and none is found to be of clinical use, indicating the difficulties of training a useful model.
149. List, M., Ebert, P. & Albrecht, F. Ten simple rules for developing usable software in computational biology. *PLoS Comput. Biol.* **13**, e1005265 (2017).
150. Sonnenburg, S. Ā., Braun, M. L., Ong, C. S. & Bengio, S. The need for open source software in machine learning. *J. Mach. Learn. Res.* **8**, 2443–2466 (2007).

Acknowledgements
The authors thank members of the UCL Bioinformatics Group for valuable discussions and comments. This work was supported by the European Research Council Advanced Grant ProCovar (project ID 695558).

Author contributions
All authors researched data for the article, contributed substantially to discussion of the content, wrote the article and reviewed the manuscript before submission.

Competing interests
The authors declare no competing interests.

Peer review information
Nature Reviews Molecular Cell Biology thanks S. Draghici who co-reviewed with T. Nguyen; B. Chain; S. Haider; F. Mahmood; and the other, anonymous, reviewer(s) for their contribution to the peer review of this work.

Publisher's note
Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

RELATED LINKS
Caret: https://topepo.github.io/caret
Colaboratory: https://research.google.com/colaboratory
Graph Nets: https://github.com/deepmind/graph_nets
MLJ: https://alan-turing-institute.github.io/MLJ.jl/stable
PyTorch: https://pytorch.org
PyTorch Geometric: https://pytorch-geometric.readthedocs.io/en/latest
scikit-learn: https://scikit-learn.org/stable
Tensorflow: https://www.tensorflow.org