Chapter 4
Primer on Machine Learning

The previous chapter introduced various concepts from statistics and probability relevant to forecasting. As many state-of-the-art approaches rely on machine learning, this chapter will introduce some fundamental definitions and concepts. It does not intend to provide an in-depth understanding but instead plans to overview the main concepts as they are relevant to this book. It provides an overview of practically relevant concepts when using software packages to fit and configure machine learning models. It does not introduce specific algorithms as those machine learning algorithms that are typically used in load forecasting are discussed in detail in Chap. 10. More in-depth overviews of the approaches can be found in the list of further reading in Appendix D.2.

4.1 Definitions and Related Concepts

A common definition of **machine learning** is that it includes algorithms that enable computers to learn from **data** and improve **performance** within a specific **task** without being explicitly programmed. A typical such task is to describe the relationship between a set of input variables that are typically measured or preset and have some influence on one or several outputs (see the next section for other machine learning tasks).

With this definition, machine learning can be distinguished from classic **algorithms** and programs studied in computer science, i.e., a finite sequence of well-defined instructions. A traditional algorithm performs a task deterministically following the steps that have been implemented by the programmer at design time, resulting in a specific performance. In contrast, a machine learning algorithm uses experience/observations, i.e. data, to improve the performance within the task, possibly even when in operation. This allows the algorithm to tackle tasks that are too difficult to solve with classical algorithms and programs written and designed by human beings. A second way of distinguishing machine learning from classical programming is that for classical programming, one provides the input and a function (the algorithm) to compute an output. In machine learning, one provides some example inputs and outputs to find a function which can then be used with new inputs to compute outputs.

The above definition can also be used to distinguish machine learning from **statistics** (see the introduction to some basic concepts in Chap. 3). One notion of distinguishing the two is that statistical models are designed for **inference** about the relationships between variables, and machine learning models are developed to make the most **accurate predictions**, i.e. to maximise performance. A common purpose of statistical models is to make inferences about the relationships between variables, i.e., creating a mathematical model of the process by which data was generated to formalise understanding or test a hypothesis about the system behaviour.

While statistical models can also make predictions, this is often achieved by making parametric assumptions, like assuming that data follows a specific distribution (see Chap. 3). While this improves model understanding and **interpretability**, it introduces **model bias** that for complex data may hinder achieving accurate predictions. Similarly, as too many input variables hinder the interpretability of the process, statisticians may want to avoid situations where $p >> n$, i.e., the number of input variables is much larger than the number of samples. Hence, a statistician may want to attempt to avoid over-parameterisation by performing a variable selection, removing terms that do not contribute significantly to the inference. This generally improves model understanding but may inhibit prediction accuracy, as even small contributions can improve predictions.

In contrast, a machine learning model aims to learn from experience (past data) with the main goal of improving a prediction, in other words it typically sacrifices understanding of the underlying mechanisms for performance. Machine learning models are mostly non-parametric methods (e.g. see Sect. 3.4) and are typically capable of handling many input variables without extensive manual preprocessing or feature selection. Utilising complex models and removing restrictive assumptions, machine learning applies optimisation techniques to find approximate algorithmic solutions (see Sect. 4.3). In contrast statistical models can sometimes find exact closed-form solutions. Alongside optimisation, machine learning methods are often concerned with developing methods to avoid **overfitting**, i.e., finding models and methods that are capable of **generalising** well to new data points that they have not been trained on. This includes different **regularisation** methods (see Sect. 8.2.5). All this is done to improve prediction performance at the cost of interpretability compared to statistical methods.

While the previous paragraphs highlighted some main differences between machine learning to classic computer algorithms and statistics, they are not completely distinct. There are considerable overlaps between the concepts, and machine learning heavily relies on classical computer science and statistics techniques and approaches. For instance, knowing the data-generating process typically provides insights into what makes a good predictor and is often an essential step in the applied modelling process. Machine learning also relies on several areas of computer science and algorithms, for instance, when datasets become too large to fit into the memory of a single computer (or its GPU). Much of practical machine learning is concerned with developing strategies to manage millions, billions and even trillions of parameters using computer science methods like distributed computing (this aspect is not addressed in this book). Beyond those, machine learning also heavily relies on other mathematical concepts such as optimisation, matrix algebra and calculus.

How is this translated to short term load forecasts?

As in many domains in recent years, machine learning models have increasingly been applied to forecasting problems. In contrast to many other disciplines like image recognition or natural language processing, for time series forecasting they have not yet dominated over other approaches in the field, like simple benchmarks and more sophisticated statistical approaches, as discussed in Chap. 9. This is because many time-series problems have limited data availability, and many machine learning algorithms are often unnecessarily complex. Only in recent years have machine learning models, for instance, recurrent neural networks or gradient boosting, started to outperform other methods consistently (see, for example, the discussion of the M5 time series forecasting competition [1]). With the advent of specialised deep learning approaches like DeepAR [2] and N-BEATS [3], there is a likely trend that more advanced specialised machine learning models may improve in many time series problems. However, as statistical models allow for a better understanding of the relationships between the available data and the forecast, an accurate statistical model should always be used as a benchmark in the load forecasting process (see Chap. 9). The M5 competition showed [1], that *combinations* of statistical and machine learning models can reach state-of-the-art results with the advantage of remaining at least partly interpretable, combining the benefits of both approaches. This makes them particularly interesting for real-world applications.

4.2 Machine Learning Taxonomy and Terms

The former section introduced **machine learning** as algorithms that enable computers to improve the performance within a specific **task** by learning from data. The most common machine learning task is to describe the relationship between a set of input and output variables. This task is called **supervised learning**. Besides supervised learning, there are other sub-types of machine learning, most importantly **unsupervised learning** and **reinforcement learning**.

4.2.1 Supervised Learning

Supervised learning is the task of learning the relationship between a set of input and output variables from data, i.e., to learn some function f to be able to make predictions. These inputs are often referred to as **instances** or **examples**. An instance is a collection of values that can be measured or obtained in some way (e.g. through sensors). It is often denoted as $\mathbf{X} \in \mathbb{R}^n$. In statistical terms, an instance is a realisation vector of the n random variables $X_1, X_2, \ldots, X_n$, so that one can also write $\mathbf{X} = (X_1, X_2, \ldots, X_n)$ (Sect. 3.1). These inputs are often called predictors or independent variables in the statistical literature. In the machine learning literature (and the remainder of this Chapter), the term **features** is more commonly used. Throughout this book, both terms will be used.

A statistician may refer to the outputs as the **response** or the **dependent variables**. Depending on if the task is to predict a numeric variable or a qualitative variable (like the membership to a **class**), in machine learning, the output is referred to as **target** or **label** in the machine learning literature. If the supervised learning task is to predict a numeric variable, the task is referred to as **regression**. To solve this, the machine learning algorithm aims to model a function $f: \mathbb{R}^n \to \mathbb{R}$, or $f: \mathbb{R}^n \to \mathbb{R}^k$ in the case of multiple target variables. In the latter case, the regression problem may be referred to as multi-target or multi-output regression. Note this is not to be confused with multiple regression or multivariate regression, which refers to the dimensionality of the inputs, i.e., $n > 1$. Note that the term regression is sometimes used to denote linear regression (see Sect. 9.3). However, linear regression is only one specific regression method, and the regression task can be performed using many different algorithms, as will be discussed in this book.

A common machine learning task is to predict the assignment of an instance to one of k categories, or **classes**. Here, the machine learning algorithm is equivalent to modelling a function $f: \mathbb{R}^n \to \{1, \ldots, k\}$, with each class assigned an integer. This output can be a numeric code representing a specific class, but more commonly, machine learning algorithms produce a probability distribution over the classes. This task is referred to as **classification**. Depending on whether the output is univariate or multivariate, the output is denoted as $Y \in \mathbb{R}$ or $\mathbf{Y} \in \mathbb{R}^k$ (for the sake of simplicity in the following the univariate case is presented as a special case of the multivariate case for $k = 1$).

The relationship f is learned from a **dataset**. While formally, the order should not matter, a dataset can be seen as a set of N tuples of instances with the respective labels or targets, i.e., $\mathcal{X} = \left\{(\mathbf{X}_1, \mathbf{Y}_1), (\mathbf{X}_2, \mathbf{Y}_2), \ldots, (\mathbf{X}_j, \mathbf{Y}_j), (\mathbf{X}_N, \mathbf{Y}_N)\right\}$. However, as many algorithms make use of linear algebra, it is also common to denote the set of inputs of the dataset as a matrix $\mathbf{X}$ and the set of output as a vector or matrix $\mathbf{Y}$. Note that the inputs have been introduced as n-dimensional vectors for simplicity, but for different machine learning tasks, they may be of higher dimensionality, e.g. for images or videos. Hence, they may be considered more generally as **tensors**. However, for convenience and clarity, the focus will be on matrices and vectors, which will be the most common form for load forecasting.

4.2.2 Unsupervised Learning

Unsupervised learning typically refers to algorithms that learn and identify patterns within the data without labels. Denote the dataset as set $\mathcal{X}$ with N members $\mathbf{X}_1, \ldots, \mathbf{X}_i, \ldots, \mathbf{X}_N$ with each $\mathbf{X}_i = (X_1, \ldots, X_j, \ldots, X_n)^T$ consisting of n features. Unsupervised learning tasks can be used to either model a pattern within the set of data $\mathcal{X}$, model within each of the instances $\mathbf{X}_i$, or both.

The type of problem that is most commonly associated with finding patterns within the dataset is the task of finding partitionings or groupings of a dataset, i.e. **clustering** a dataset. The goal is to find k groupings $\mathcal{X}'_1, \ldots, \mathcal{X}'_i, \mathcal{X}'_k$ such that $\cup_{i=1}^{k} \mathcal{X}'_i = \mathcal{X}$. The most common algorithms are the centroid-based **k-means clustering**, distance-based **hierarchical clustering** and the density-based models **DBSCAN** and **OPTICS**. A discussion of those algorithms is not part of this book, but see [4] for k-means and hierarchical clustering and [5] for a discussion on DBSCAN and OPTICS. An example of another approach which can be used for clustering, called **finite mixture models** is given in Sect. 11.3.2, except in this case it is used to model complex distributions.

The most common reason for finding patterns within each of the individual instances $\mathbf{X}_i$ is **dimensionality reduction** and the related problem of finding **embeddings**, i.e., meaningful **latent** representations of the data. Dimensionality reduction methods aim to address the curse of dimensionality[1] by finding lower-dimensional representations of the data. This lower-dimensional representation of the data can be used as features, for instance, in supervised learning or forecasting. A model that uses lower-dimensional data as input to make the predictions may be referred to as a **down-stream model**. It can also be useful for visualising high-dimensional data in 2D or 3D representations. Popular methods are principal component analysis (PCA) and t-Distributed Stochastic Neighbor Embedding (t-SNE). Figure 4.1 shows the result of using PCA to reduce the time series of three households from 96 values per day (15-min data) into two dimensions. On the left, there are some example samples of three households in 15-min resolution. One can see that the behaviour is generally quite different with load at different times of the day. On the right is a scatter plot of the two-dimensional data after applying PCA with the goal of finding two descriptive features of the data. The colour highlighting of the specific households is added to illustrate how even though the dimensionality has been drastically reduced, the data of the different households generally stay together, indicating that some differences between them are preserved in this low dimensional representation.

A detailed discussion of these methods is not within the scope of this book. Finding a suitable latent representation of raw inputs is essential in working with image or text data. For instance, many machine learning models need fixed-length numerical input sequences and cannot handle sentences of different lengths. It is commonly

[1] In short, the curse of dimensionality is the rapid increase in observations that are required for accurately estimating relationships as the number/dimension of features in the inputs increases. To illustrate this note how points in 2-dimensional space are much less isolated than points in 3-dimensions.

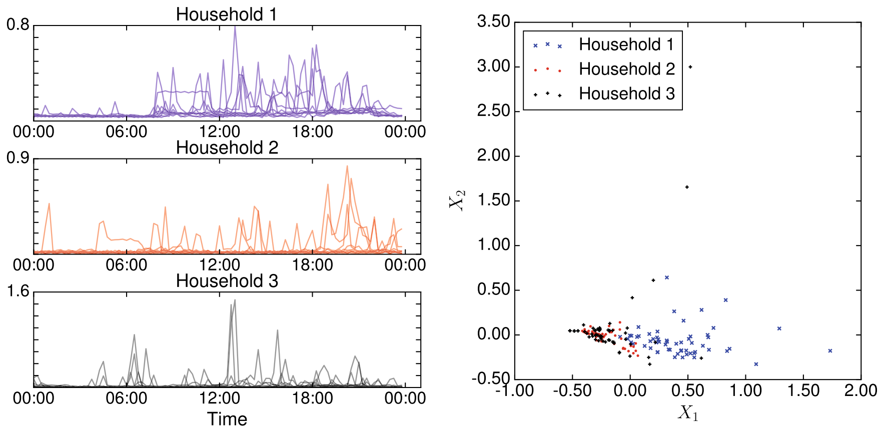

Fig. 4.1 Samples of a dataset of household-data in 15-min resolutions (left) and the resulting two-dimensional representation after applying PCA (right)

associated with neural networks (Sect. 10.4). The goal is to find a lower-dimensional representation of a fixed width, also for different length inputs (e.g., of different sentence lengths) in a latent space where similar instances are closer together.

Density estimation can also be considered an unsupervised learning task and learns patterns within both the dataset and the instances. Section 3.4 introduces some more classical approaches to density estimation from the statistics domain. However, more recently, for complex distributions like text, images and video, **generative machine learning** methods like variational auto-encoders (VAE), normalising flows, generative adversarial networks (GAN), and diffusion models have been introduced. Some of these concepts are sometimes applied to time series, including load data, but are not well established yet in this domain and are hence not further discussed in this book. The goal of the model is to estimate a distribution (see Sect. 3.1), e.g., to make inferences about uncertainties in the forecast estimate, or it can also be used to sample new data (or realisations) from the distribution of the dataset.

More recently, additionally semi-supervised and self-supervised learning methods have emerged, for instance, as part of large language models. Here, either labels partly exist to train embedding models, or they are created by partially obscuring parts of the data. Latent representations are also the core of generative models. However, those are not well established yet in this domain and are hence not further discussed in this book.

4.2.3 Reinforcement Learning

The third common machine learning sub-category is reinforcement learning. In contrast to unsupervised and supervised learning algorithms, it contains algorithms that do not learn from a fixed dataset but interact with the environment, i.e., a feedback loop between the learning system and data from the environment. The reinforcement learning task is typically modelled as a **Markov decision process (MDP)**, where the learning system is denoted as an **agent** that interacts with an **environment**. First, it observes the current **state** of the environment and chooses an appropriate **action** based on a learned **policy**. The action results in a **reward** as feedback from the environment. Then the environment is in a new state, and the process continues in an iterative manner. The policy is adjusted based on the rewards so that the agent learns. Depending on the algorithms, the agent chooses actions that are known to work well (**exploitation**), or it may explore new actions (**exploration**). As the policy maps state/action pairs to rewards, this mapping can be supported by supervised learning algorithms and, more recently, deep learning algorithms (Sect. 10.5).

However, often real-world applications suffer from different challenges. One problem is the **credit assignment problem**, where often the reward signal is delayed and only available after a sequence of actions, making it difficult to attribute the influence of individual actions to the reward. A second problem is **reward hacking**, where misformulated objective functions can lead to unexpected results. Lastly, the exploitation of new actions may often not be desirable in an application where the cost of ineffective actions is high. Here, simulations (often referred to as **gyms**) can support learning an optimal policy. Reinforcement learning concepts have also been partially applied to the time series forecasting problem, but they are not well established yet and are not further discussed in this book.

How is this translated to short term load forecasts?

Time series forecasting can be formulated as a multivariate regression problem. Therefore supervised machine learning models can generally be applied to time series forecasting problems. While supervised approaches are the most common, unsupervised and reinforcement learning methods can be used in forecasting, either on their own or, more commonly, in combination with supervised approaches. For instance, when forecasting an aggregated time series, clustering techniques can be used to partition the data of the time series that make up the signal. Models are then trained on the resulting clusters before combining the predictions. This can produce improved accuracy compared to directly forecasting the aggregated time series since individual series may have similar features which can be used to design an accurate forecast model of the cluster. Further, dimensionality reduction can improve forecasting models and be applied in exploratory data analysis to identify important patterns and/or develop new inputs.

4.3 Introduction to Optimisation with Gradient Descent

As described above, when using machine learning, the optimal parameters have to be determined based on the prediction errors on a training set. For instance, in neural network models (Sect. 10.4), the weights of the networks must be trained. However, besides the prediction error, other components affected by the model must be optimised at the same time, for instance, the range (or constraints) of the chosen parameters (see Sect. 8.2.4 on regularisation). Overall, this function to be optimised is called the **loss or cost function**.

Whereas optimisation problems in many of the statistical methods can be solved using an analytical **closed-form solution**, most optimisation problems in machine learning (like finding the weights of a neural network) are rather complex, e.g., having non-convex cost functions, i.e., they are having multiple **local optima** or **saddle points**. Figure 4.2 gives an example of such a loss function of a 56-layer pre-trained convolutional neural network. For simple cost functions, a **global optimum** can often be found, but for more complex cost functions, the solution will often be only a local optimum. Hence optimisation is often not deterministic, in other words running the same procedure (e.g. the training of a neural network) can result in different solutions, i.e., models that perform better or worse.

Most state-of-the-art machine learning models, especially neural networks, use some form of gradient descent. The **parameters**, i.e., the weights $\boldsymbol{\beta}$, are adjusted

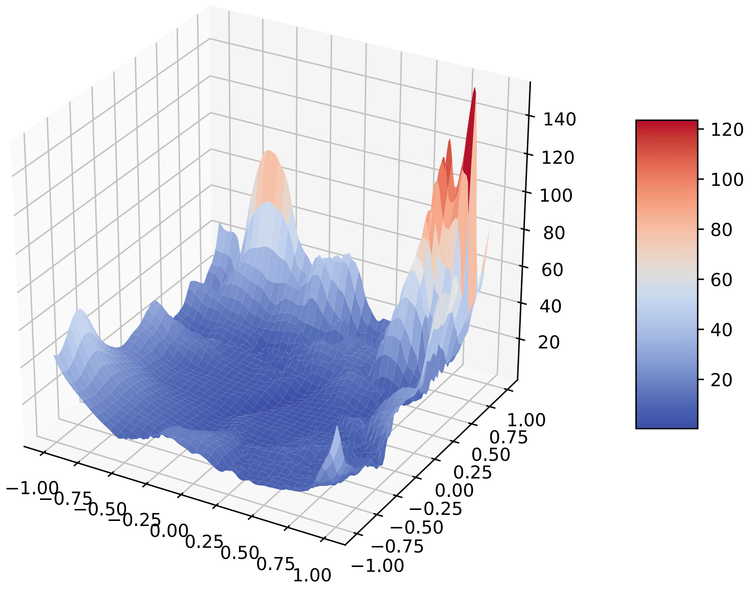

Fig. 4.2 Example loss function along two random normalised directions of pretrained 56-layer convolutional neural network ResNET-56 for image recognition, created with Loss Landscape tool described in [6]

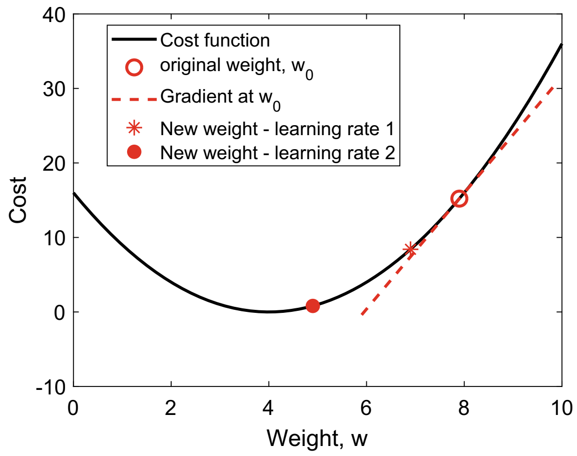

Fig. 4.3 Illustration of gradient descent for a basic cost function. The value w is updated based on the direction in which the gradient descends. The length the update moves is based on the learning rate. Two different learning rates are illustrated here

according to the gradient of the loss function (see the introduction on artificial neural networks in Sect. 10.4). Hence, if the gradient of the cost function is positive with respect to some current weight, it means the cost will decrease if the weight decreases. Similarly, if the gradient is negative then the cost function will decrease if the weight increases. It should be noted that many statistical approaches also use gradient methods, especially for problems with large numbers of parameters and observations where finding the optimal directly in a closed-form is difficult.

To demonstrate this process consider a basic cost function (defined by $(w - 4)^2$) as shown in Fig. 4.3. Initially, the weight is $w = 7.9$, and the cost function has a positive gradient at this point. This shows that local to this weight, increasing the weight will increase the cost function. Hence, the weight should be reduced to reduce the value of the cost function and find a value closer to the optimal (zero in this case with $w = 4$). The process is then repeated at the new value. The question remains of how much to reduce the value. This is determined by the so-called **learning rate** or **step-size**. Two different learning rates are shown in Fig. 4.3 where one gets closer to the true minimum (at $w = 4$) for a single step compared to the other rate after one iteration. Clearly, the learning rate is an important hyper-parameter for the algorithm (see Sect. 8.2.3 on tuning hyper-parameters). Too small, and the convergence will take too long (and potentially be stopped too early) to reach the minimum. Too big, and the solution will be too unstable and potentially not converge at all.

In this simple example, the gradient was determined for the full training data. This is called **batch gradient descent**. It calculates the error for each instance using efficient matrix operations and takes an average over the whole dataset to determine the gradient. This averaging provides a stable learning path and hence leads to quick convergence. This is only feasible for simple problems and small datasets. For larger neural networks, for instance, convolutional neural networks and large datasets (e.g. images), this is not feasible as the gradient calculation becomes too computationally complex and the matrices will not fit into your computer's memory (although even

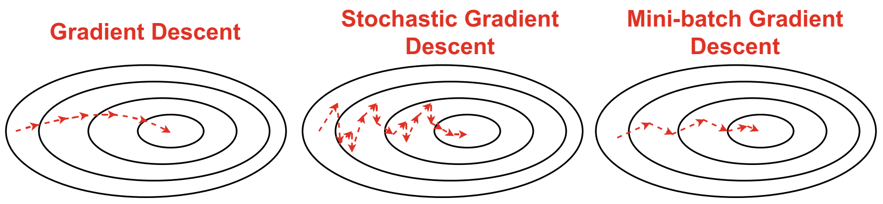

Fig. 4.4 Illustration of the convergence path of gradient descent (smoothly), stochastic gradient descent (unstable) and mini-batch gradient descent (compromise)

if you had large amounts of memory this is perhaps an inefficient use of resources). Instead, on large datasets, using samples from the dataset can often be enough to produce a good approximation of the current iteration's gradient.

When only one instance is considered at each step, this is called **stochastic gradient descent**. The advantage of stochastic gradient descent is that the algorithm is much faster at every iteration. However, the algorithm results in a less regular and stable learning path compared to batch gradient descent. Instead of decreasing smoothly, the cost function will "zig-zag" and may even temporarily move "back up the hill", as individual samples do not accurately approximate the entire dataset's gradient. While this may seem like an undesirable property, this is helpful in training complex neural networks as it introduces a randomness to the optimisation procedure that can help the estimate escape local minima or saddle points. Hence, most current deep learning models use a variant of **mini-batch gradient descent** that combines the concepts of batch and stochastic gradient descent. Figure 4.4 illustrates the paths of these three gradient descent algorithms.

In mini-batch gradient descent, the algorithm computes the gradient based on a subset of the training set at each step instead of the complete dataset or only individual instances. This provides a trade-off, as it takes advantage of efficient matrix operations during the gradient calculation, resulting in a smoother and more stable convergence than stochastic gradient descent. One downside is that this introduces additional **hyperparameters** to the training process, as besides the step-size, a **batch-size** also has to be provided. The mini-batch size is chosen to ensure enough diversity to escape local minima while providing some stability and enough computational efficiency from fast matrix calculations.

Besides the way the training set is split up to calculate the gradient, there are many other ways that the vanilla gradient descent can be improved. One popular modification is to add **momentum**. Regular stochastic gradient descent may have trouble in areas of the loss function where the surface gradient is much steeper in one dimension than in another. Here, a momentum term is added that increases for dimensions where the gradients point in the same directions and decreases for dimensions where gradients change directions. This results in faster convergence and fewer oscillations when moving towards a local optimum (see Fig. 4.5 for an illustration).

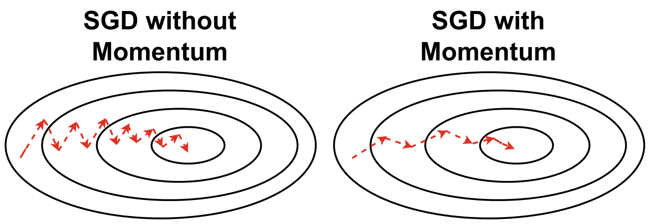

Fig. 4.5 Illustration of the convergence path of stochastic gradient descent without momentum and with momentum

Many optimisation algorithms introduce some form of **adaptive learning rate** that changes the step size value according to some rule. Simple schedules may **decay** (reduce) the step size over time to produce larger changes at the beginning of the training process and then fine-tune towards the end when moving towards a local optimum. A variant is a **cycling learning rate** where the learning rate iteratively becomes larger and smaller according to a rule to improve the chances of escaping local optima. An overview of the many existing optimisation algorithms is out of the scope of this book. An overview of some popular optimisers is given in [7]. One effective and popular optimisation algorithm is Adaptive Moment Estimation (Adam), which uses an adaptive learning rate and the ideas behind momentum. It will usually converge faster to a local minimum than using vanilla stochastic gradient descent without momentum with a simple learning rate decay schedule. Further, it is less prone to get stuck in saddle points and relies less on initialisation parameters like step size and decaying schedule. Hence, Adam with default hyper-parameters is a decent method to use in practice for training deep neural networks.

4.4 Questions

For the questions which require using real demand data, try using some of the data as listed in Appendix D.4.

1. Select the time series of an individual household or building and arrange your dataset in a matrix so that each row represents one day. Resample it to hourly data so that the matrix has 24 columns. Apply principle component analysis (PCA) using a library of your preference, for instance, Scikit-Learn[2] to reduce the dimensions. Then also, apply t-SNE for comparison. Visualise the results for two dimensions and three dimensions using scatter plots. Do you see clusters of data points? Find explanations for what could cause different groupings. Verify your hypothesis by using different colouring in the plot. Next, repeat the application of each of the algorithms. Do you see variation in the results?

[2] https://scikit-learn.org/stable/index.html.

2. Use a load dataset that contains data from several households. Create a matrix $\mathbf{X_1}$ by rearranging the data so that each row of the matrix represents one day of one household. Run k-means clustering with the objective of finding 10 clusters. Visualise members of each of the clusters and compare them. Do you see distinct clusters or are their clusters with very similar patterns that could be merged? Try less target clusters. Does this result in "better" groupings? In the absence of a "ground truth", what could generally be ways to assess and compare the results in clustering?
3. Clustering can be done using the raw load data or by modelling features from the load data. $\mathbf{X_1}$ contains the raw data already. Next, create a matrix $\mathbf{X_2}$ by applying PCA to reduce the dimensionality of the data to 6 components (i.e., the matrix has 6 columns). Create a third matrix $\mathbf{X_3}$ where you derive some manual features, such as the mean and max of certain times of the day. Repeat the k-means clustering in the previous question with the target of 10 clusters. Visualise cluster members of each cluster again and compare the results across the 3 representations (raw, manual feature and automated features from PCA). Do the results vary much? Does one method result in more distinct clusters?

References

1. S. Makridakis, E. Spiliotis, V. Assimakopoulos, The M5 accuracy competition: results, findings and conclusions. Int. J. Forecast. (2020)
2. David Salinas, Valentin Flunkert, Jan Gasthaus, Tim Januschowski, Deepar: probabilistic forecasting with autoregressive recurrent networks. Int. J. Forecast. **36**(3), 1181–1191 (2020)
3. B.N. Oreshkin, D. Carpov, N. Chapados, Y. Bengio, N-beats: neural basis expansion analysis for interpretable time series forecasting (2019). arXiv:1905.10437
4. T. Hastie, R. Tibshirani, J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer Series in Statistics (2009)
5. M. Ankerst, M.M. Breunig, H.-P. Kriegel, J. Sander, Optics: ordering points to identify the clustering structure. ACM Sigmod Record **28**(2), 49–60 (1999)
6. H. Li, Z. Xu, G. Taylor, C. Studer, T. Goldstein, Visualizing the loss landscape of neural nets, in *Neural Information Processing Systems* (2018)
7. S. Ruder, An overview of gradient descent optimization algorithms (2017). arXiv:1609.04747v2