CHAPTER

9

Deep neural networks

Sandra Vieira[1], *Walter Hugo Lopez Pinaya*[1,2], *Rafael Garcia-Dias*[1], *Andrea Mechelli*[1]

[1] Department of Psychosis Studies, Institute of Psychiatry, Psychology & Neuroscience, King's College London, London, United Kingdom; [2] Centre of Mathematics, Computation, and Cognition, Universidade Federal do ABC, Santo André, São Paulo, Brazil

9.1 Introduction

Over the last decade, several different machine learning methods have been applied in brain disorders. The most popular methods include linear regression, logistic regression, Support Vector Machine (SVM), and Support Vector Regression (Chapters 4–7). Their popularity is not surprising in light of their simplicity, interpretability, and ease of use (Wolfers et al., 2015). Common to all these methods, however, is the inability to deal with data in their raw form. For this reason, much of the effort that goes into developing a successful machine learning pipeline is spent carefully creating useful features from the raw data through a process known as *feature engineering* (Domingos, 2012).

But what if instead of manually crafting features, we could learn them directly from the data? This is the main premise of representation learning, a class of machine learning methods that tries to automatically find the optimal set of features, i.e., representation, from the data. Deep learning (DL) is one family of representation learning methods that is capable of detecting multiple levels of representation. This is achieved by combining consecutive layers of simple nonlinear transformations, which results in increasingly abstract features (Goodfellow, Bengio, & Courville, 2016). Let us imagine an example of a facial recognition task. The input data come in the form of an array of pixel values (an image). The first layer of representations typically captures low-level features such as the presence or absence of edges. The second layer builds on these representations by learning particular motifs such as different arrangements of edges. The third layer assembles motifs into larger combinations that correspond to parts of a face (corner of the eyes, tip of the nose, etc.). Finally, subsequent layers detect objects (eyes, mouth, nose, etc.) as combinations of these parts (LeCun, Bengio, & Hinton, 2015). This layer-wise structure can learn very intricate functions capable of simultaneously distinguishing between very similar faces while ignoring large irrelevant aspects of the images such as the background, lighting, or angle of the face. It is this automatic learning of hierarchical representations that makes DL fundamentally different from the conventional "shallow" (as opposed to "deep") machine learning methods discussed in earlier chapters.

The history of DL can be traced back to the perceptron, one of the first attempts to model the biological neuron (McCulloch & Pitts, 1943). After several setbacks (for a detailed account, see Schmidhuber (2015)), this pioneering work resulted in the development of what is now known as artificial neural networks, i.e., networks comprised of perceptron-like units. However, a key limitation of such networks was that they were able to handle only a limited number of layers. It was just in the 2000s that researchers developed a new approach for training artificial neural networks with several hidden layers which allowed for greater levels of complexity and abstraction (we will revisit this breakthrough in Chapter 11) (Hinton, Osindero, & Teh, 2006). This progress led to the development of a new family of machine learning methods now known as *deep learning*. Since then, DL has been found to outperform previous state-of-the-art classification methods in areas such as speech recognition, computer vision, and natural language processing (Krizhevsky, Sutskever, & Hinton, 2012; Le, 2013; LeCun et al., 2015) and has become one of the most promising approaches in machine learning.

It is important to recognize that DL includes a vast number of different types of models, each one with different purposes, such as autoencoders, convolutional neural networks, recurrent neural networks, generative adversarial networks, among others. In the first part of this chapter, we focus on the simplest DL model—a supervised learning model known as *deep neural network* (DNN)—to introduce the fundamental concepts of DL, namely its architecture and how learning takes place. Several terms can be used to refer to a DNN, including multilayer network, multilayer perceptron, fully connected neural network, and similar variations. For consistency, the remainder of this chapter uses the term deep neural network or DNN. Although promising, DL is not a magic bullet. Therefore, it is necessary to acknowledge its limitations to avoid spurious interpretations. Thus, the second part of this chapter focuses on the main challenges of DNNs, which are also relevant to other DL models. In the last part, we focus on recent applications of DNNs to brain disorders and discuss three studies in detail. In the next chapters, we cover two additional DL methods that are becoming increasingly popular: convolutional neural networks and autoencoders.

9.2 Method description

9.2.1 Structure

DNNs are organized in a layer-wise structure (Fig. 9.1). Each layer comprises a set of neurons (also known as units or artificial neurons) (Fig. 9.1A). Each neuron is connected to the neurons in adjacent layers to create a network (Fig. 9.1B). Connections are represented by weights that reflect the strength and direction (excitatory or inhibitory) between two neurons, much like a synapse between two biological neurons. Each neuron transforms the incoming data by calculating the weighted sum of the output of the neurons in the previous layer and then passing it

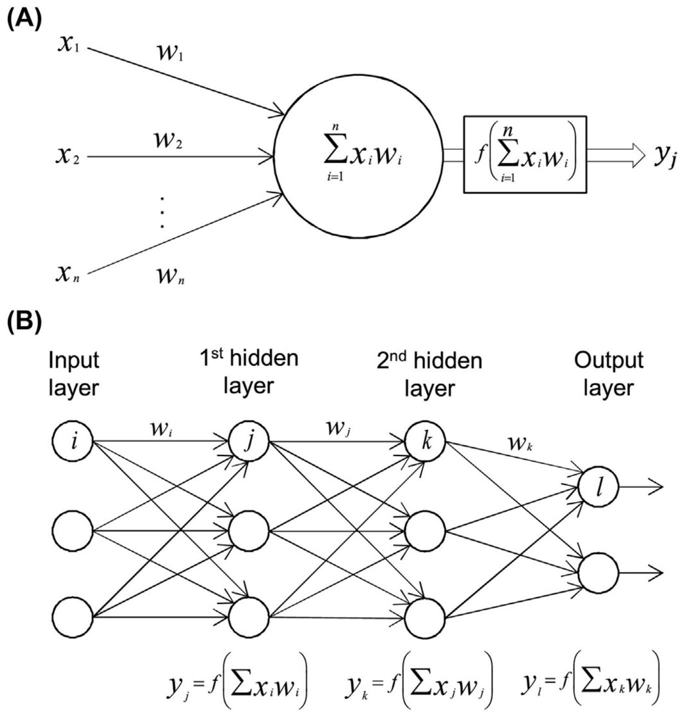

FIGURE 9.1 Deep neural network's structure. (A) At every neuron of the input layer, each input x_i has an associated weight w_i. The sum of all weighted inputs, $\sum x_i w_i$, is then passed through a nonlinear activation function f to transform the preactivation level of the neuron to an output y_i (the bias terms have been omitted for simplicity). The output y_j then serves as input to a neuron in the next layer. There are several activation functions one can use. One of the most commonly used in modern deep learning is the rectifier linear function (ReLU). (B) The information is thus propagated through the network, from one layer to the next, until it reaches the last layer, where the predicted score is outputted (process known as forward propagation). *Adapted from Vieira, S., Pinaya,W. H. L., & Mechelli, A. (2017). Using deep learning to investigate the neuroimaging correlates of psychiatric and neurological disorders: Methods and applications.* Neuroscience and Biobehavioral Reviews, 74, *58–75. https://doi.org/10.1016/j.neurobiorev.2017.01.002.*

through a nonlinear function to derive the output for that neuron. There are several nonlinear functions, also known as *activation functions*, which can be used; some of the most common ones include the rectified linear unit, hyperbolic tangent activation, or the sigmoid function.

In a DNN, the first layer corresponds to the input layer where the data are entered into the model. In neuroimaging, the input data can be represented as a one-dimensional vector with each value corresponding to the gray matter volume of a brain region, for example. The last layer is the output layer. In a classification task, the number of neurons in the output layer corresponds to the number of classes. Here, a special nonlinear function is typically used to yield the probability of a given subject belonging to each class; this nonlinear function is called *softmax* function. In the investigation of brain disorders, the output layer could generate the likelihood of a given subject belonging to the patient and the control group, for example. The layers between the input layer and the output layer are referred to as hidden layers, and their number represents the depth of the model (hence the term "deep" learning). The consecutive nonlinear transformations and propagation of information form the input through the hidden layers until it reaches the output layer, which is known as *forward propagation* (Fig. 9.1B).

A key characteristic of DNNs is that each neuron connects to all neurons in the next layer. For this reason, this type of network is also referred to as "fully connected" DNN. This fully connected structure has important implications for deciding whether a DNN is a suitable architecture for a given type of feature set. In high-dimensional data, such as neuroimaging or genetic data, assigning an input neuron to each voxel or single-nucleotide polymorphisms could easily result in millions, if not billions, of weights to estimate, which would not be feasible to compute. Therefore, DNNs are more suitable for lower-dimensional data. In the next chapter, we discuss how an additional architecture—convolutional neural network—is better suited for high-dimensional data.

When building a DNN, one has to choose the number of layers and neurons that are part of the network. Unfortunately, there is no established way of selecting the optimal number of layers and neurons a priori. One way to deal with these hyperparameters is through *hyperparameter tuning*, discussed in Chapter 2 and Section 9.2.3.4 of this chapter.

9.2.2 Training

In a typical DNN, training consists of an iterative process of adjustment of the weights between the neurons within the network, much like a human brain learns through the fine-tuning of connections between neurons (Bengio, 2009). These weights can be thought of as "knobs" that determine the relationship between the input data and the network's output (LeCun et al., 2015). The main aim of training is to tweak these "knobs" in such a way that maximizes the strength of the relationship between input and output. The question is then: how do we know how to tweak the values of the weights? The trick is to think of training as an optimization problem: find the network's weights that minimize the difference between prediction and true target (i.e., error). The error can be measured with a *loss function*, such that the smaller the value of the loss function, the closer the predictions are to the target variable. There are several ways in which this optimization can be implemented. The most commonly used are gradient descendant (GD)−based optimizers (such as vanilla GD or Adam Kingma & Ba, 2014). Below we provide a brief introduction to GD (a detailed explanation is outside the scope of a nontechnical introduction to DL; for a detailed description, see Goodfellow et al., 2016).

At the beginning of the training phase, the weights of the DNN are initialized at random. Once a training observation (e.g., data from one participant) has been entered into the input layer, the information is forward propagated through the network until it reaches the output layer, where the network outputs a prediction value. Here, the error is calculated using the loss function, followed by the estimation of how much we need to alter the network's weights to minimize the value of the loss function. The exact values of these changes are obtained using an element from calculus called *gradient*. A gradient measures how much the output of a function changes if one changes its variables by a certain amount. Therefore, it is used to indicate the slope of the loss function for the current weights. So, using the gradient, we can get the direction we need to change the weights to go down the slope (hence the name gradient descendant) to minimize the loss function. Now that we have found the direction, we need to discover how much to nudge the weights. Besides the direction, the gradient has a magnitude that indicates how steep the slope is. This magnitude is multiplied by a hyperparameter called learning rate to finally obtain how much to change the weights. Finally, the weights are updated, and the process is repeated. Note that it is not a trivial task to calculate the gradient of the weights of the network's hidden layers. It was only in the 1980s that an efficient method of computing gradients was developed, which is known as "backward propagation of errors" or *backpropagation*. The development of backpropagation made it possible to efficiently train neural networks that had additional layers between the input and output layers.

An intuitive way to think about GD for optimizing DNNs is to first imagine the loss function as a chain of mountains and valleys, in which every point along the chain is associated with possible values of the network's weights, and where the deeper the valley, the smaller the error; this is usually referred to as the *loss landscape* (Fig. 9.2). In turn, optimization through GD can be thought of as a way to navigate blindfolded through the landscape to the bottom of the deepest valley, also known as *global minima*. Imagine you start a random point (recall that the weights

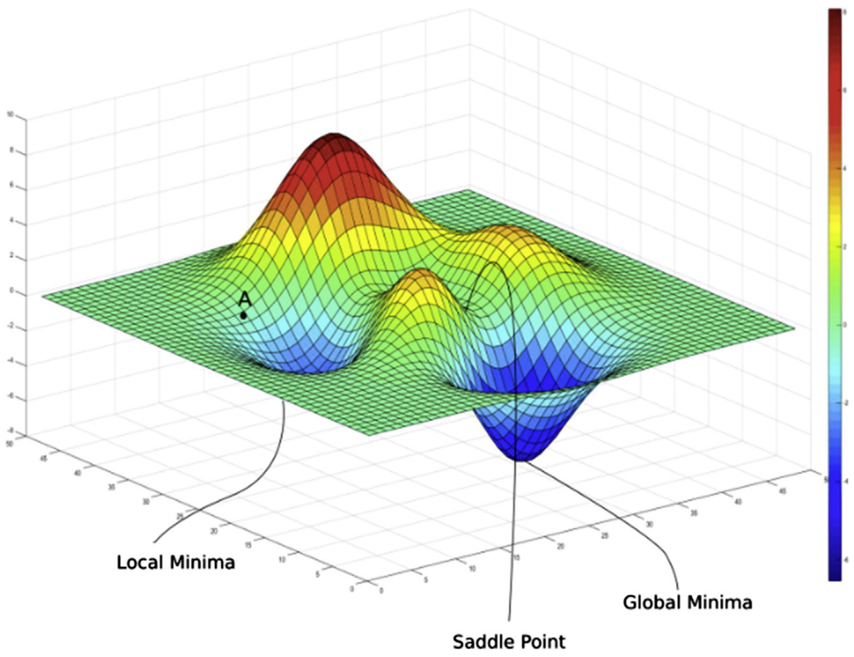

FIGURE 9.2 Loss landscape. The random initiation of the deep neural network results in an error of magnitude equivalent to a random point in the landscape (point A). The lower the "valley," the smaller the error of the loss function. Ideally, at the end of training, the loss function will have reached the global minima, i.e., the model has a very small error.

are randomly initiated) in the loss landscape (point A in Fig. 9.2), blindfolded. The first thing you need to do is to check out of all possible options, which direction results in the steepest decline, i.e., a larger decrease in the value of the loss function; this is formally known as the *gradient*: its direction reflects the direction with the steepest descent, and its magnitude indicates how steep the descent is. Once you know which direction to take, you need to decide the size of the step; this is known as *learning rate*. Once you have determined the gradient and learning rate, you are ready to take your first step. Once at your new position, you recompute the gradient and take another step. As you approach the minima, the contour of the function, i.e., the ridges of the valley, becomes almost flat. This will result in a very small gradient, i.e., there is no more "downward" to go from here, and you have reached a minimum. Now, this assumes that you took a steady step size—the learning rate was always the same. However, while you may want to take larger steps at the beginning, this might result in overshooting the minima once you get closer to it. Therefore, you can decrease the size of steps as you walk down to avoid this; how much you decrease your steps by is another hyperparameter of GD, and it is known as *learning rate decay*.

The type of GD can vary depending on the number of training samples used to calculate the error. There are three main variations of the GD: batch, stochastic, and minibatch. In stochastic gradient descent, the error and respective updates are computed for each observation in the training dataset. Perhaps the most intuitive of the GD variations, it can result in a noisy gradient signal, making it hard for the algorithm to settle on a minimum value. Batch gradient descent on the other hand considers the error of all training samples before updating the weights. Here, the more stable gradient signal may converge prematurely, thus resulting in a less optimal network's weights. Finally, minibatch gradient descent splits the training data into small batches—minibatches—that are used to calculate the error and update the model coefficients. This approach represents a reasonable trade-off between the two other methods, and as such it is the most common form of GD. Similarly to the other hyperparameters, batch size also needs to be determined a priori or tuned as part of the machine learning pipeline.

9.2.3 Main challenges

9.2.3.1 Starting weights, multiple local minima, and saddle points

DNNs are typically initialized with random weights. This has important consequences for the learning process as it implies that, depending on the value of these starting weights, the network will start at different positions in the loss landscape, which means that the optimization may get stuck in a valley, known as *local minima*, which is not the deepest point, known as *global minima*. Despite this challenge, many believe that most local minima do not differ substantially from the global minima and therefore are associated with low error (Choromanska, Henaff, Mathieu, Arous, & LeCun, 2015; Dauphin et al., 2014a; Goodfellow et al., 2016); it follows that a successful application of DL may not necessarily require finding the global minima (Bishop, 2006). The usual approach involves trying different hyperparameters involved in the GD algorithm (e.g., learning rate) and choosing the best one, i.e., the one associated with the lowest error. The nonconvexity of the cost function is an essential property to be aware of when optimizing a DNN, as it helps to understand the models' behavior and inform consequent parameter tuning. For example, with very large learning rate, you might overshoot the minima and keep bouncing along the ridges of the valley; if too small, you might take an unfeasible amount of time to get to the global minima or get stuck in a valley that is not the deepest point, known as *local minima* (Fig. 9.3).

In addition to multiple local minima, there is also the issue of saddle points. Imagine you are back at the mountain chain, blindfolded, looking for the deepest descent and reach a point from which there is a steep descend in one direction and steep ascend in another (perpendicular) direction (see Fig. 9.2); GD would keep oscillating in this region, which

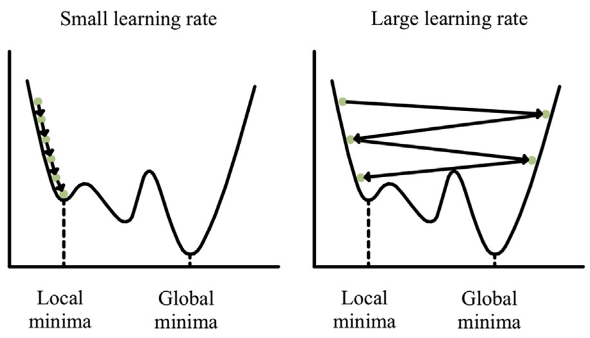

FIGURE 9.3 Effect of learning rate on weight optimization in gradient descent. (A) With a very small learning rate, the optimization is likely to reach local minima and (B) with a very large learning rate, the optimization may miss the global minima entirely.

will give the impression that it has reached a minima. Luckily there are a few approaches to address this issue; please refer to Choromanska et al. (2015) and Dauphin et al. (2014) for a detailed discussion.

9.2.3.2 *Overfitting*

Due to the connection between neurons, the training of DNNs involves the estimation of a considerable number of parameters, i.e., weights. This can lead to the model learning particular fluctuations in the training data that only work in the training set—an issue known as *overfitting* (see Chapter 2). When this happens, the model will perform well on the training data but not on unseen or test data. Fortunately, there are some strategies, collectively known as regularization, which can be used to minimize this risk. A first strategy involves the use of weight decays (e.g., L_1 and L_2 norms) to penalize models with very high weights. By forcing weights to remain low (recall that the value of the weights reflects importance), the network becomes less dependent on the training data (i.e., performance does not rely heavily on a particular set of weights) and can better generalize to unseen data (Nowlan & Hinton, 1992). A second strategy, known as dropout, consists of temporarily removing a random number of neurons and their respective incoming and outgoing connections from the network during training. This results in the extraction of different sets of features that can independently produce a useful output, which in turn has the effect of enhancing generalizability (Srivastava, Hinton, Krizhevsky, Sutskever, & Salakhutdinov, 2014). An additional approach is to stop training when the error in the training set stops decreasing, especially if this is accompanied by an increase in the error in the validation set, a strong indicator of overfitting; this approach is known as early stopping (Prechelt, 1998).

9.2.3.3 *Interpretability*

DL is often criticized for its lack of transparency and is sometimes referred to as a "black box" in contrast with other techniques, such as logistic regression, which are less complex and generate more intuitive outputs (Alain & Bengio, 2016; Yosinski, Clune, Nguyen, Fuchs, & Lipson, 2015). This lack of transparency is partially because of its reliance on multiple nonlinear transformations. Because of such nonlinearities, it can be challenging to trace the consecutive layers of weights back to the input data to identify which features are providing the most significant contribution (Suk, Lee, & Shen, 2015). Therefore, it can be difficult to assess whether the features driving prediction are clinically relevant aspects of the data (e.g., gray matter reductions in the case of structural neuroimaging) as opposed to spurious differences (e.g., movement-related artifacts). Also, a model with excellent performance may still be of limited clinical utility, for example, concerning treatment development and optimization. It is therefore unsurprising that, in recent years, the issue of how to enhance the interpretability of DL has become an expanding area of research (Grün, Rupprecht, Navab, & Tombari, 2016; Montavon, Samek, & Müller, 2018; Samek, Binder, Montavon, Lapuschkin, & Müller, 2017; Simonyan, Vedaldi, & Zisserman, 2013; Yosinski et al., 2015). Within this area, a number of studies using DL to investigate brain disorders have attempted to extract the information on the most important features driving prediction using a variety of different approaches (Deshpande, Wang, Rangaprakash, & Wilamowski, 2015; Kim, Calhoun, Shim, & Lee, 2016; Liu et al., 2014; Suk, Wee, Lee, & Shen, 2016).

9.2.3.4 *Hyperparameter tuning*

As we saw earlier in this chapter, the number of layers, the number of neurons within each layer, the activation function, the optimizer, the learning rate, and the regularization strategy are only a few examples of a long list of hyperparameters one has to consider when building and optimizing a DL model. This long list results in an endless number of possible combinations of hyperparameters. Although automated tuning, in which the algorithm is instructed to test alternative values and select those that provide the best result, is possible, it can be computationally expensive. However, with the fast-growing availability of graphical processing units, the application of DL is likely to become less expensive and more feasible in the future. Manual optimization, i.e., trial and error, is arguably the most common type of hyperparameter tuning in brain disorders research. Although guidelines are available (Bengio, 2012), manual adjustment is mostly based on the intuition of how the network behaves with different hyperparameters. This approach has the advantage of being much less computationally expensive than automatic tuning; however, it requires a great deal of technical expertise and is potentially prone to subjective bias. Importantly, regardless of the tuning method, any study that aims to be replicable should ideally report both the explored set of hyperparameters and the hyperparameters adopted in the final solution. Without the information on the architecture of the DNN and the explored and adopted hyperparameters, any result would be challenging to replicate and as such should be taken with caution.

9.3 Applications to brain disorders

In light of its ability to learn latent and abstract patterns, it has been suggested that DL may be particularly suitable to uncover complex effects within a certain modality, for example, subtle, widespread, and heterogenous reductions in gray matter volume, or even capture cross-modality relations, such as the interaction between genetics and neuroimaging (Calhoun & Sui, 2016; Plis et al., 2014; Schnack, 2017). Plis et al. (2014) is often referred to as the first study to apply DL in brain disorders, specifically schizophrenia and Huntington's disease. Since then, several other studies have applied some form of DL to a wide range of brain disorders with promising results (Vieira, Pinaya, & Mechelli, 2017). Most of these studies have used sophisticated approaches such as autoencoders or convolutional neural networks, which are described in the next two chapters. In contrast, studies using DNNs were less common. As discussed earlier, DNNs are more suitable for lower-dimensional data (see Section 9.2.2). This fact explains why, in the vast majority of studies, the input data were functional connectivity (FC) matrices, patches of voxels showing an effect of interest, or gray matter volumes from predefined regions. Also, all networks from these examples were initialized with prelearned weights, i.e., the weights were not set to a random value at the beginning of training; they were prelearned via autoencoders. This process is explained in Chapter 11. Below we review three fundamental applications of DNNs which illustrate the potential and flexibility of this approach to identify individuals diagnosed with schizophrenia and to detect the early stages of Alzheimer's disease (AD) and autism spectrum disorder (ASD).

9.3.1 Diagnostic classification in schizophrenia

In one of the first applications of DL to schizophrenia, Kim et al. (2016) used a DNN to examine whether brain activity at rest (measured with resting state functional magnetic resonance imaging) could be used to distinguish patients with schizophrenia from healthy controls. A total of 50 patients and 50 controls were included. To prepare the data for the classifier, a FC matrix was extracted from each individual's functional images. Briefly, this involved parcellating each image into 116 brain regions and calculating the whole-brain FC matrix by estimating the Pearson's correlation coefficient between each possible pairs of regions. The resulting matrix, representing the FC between a total of 6670 pairs of brain regions, was used as input data for the DNN. As we saw earlier, training a DNN requires the specification of several hyperparameters, which can be done either manually or automatically. In this investigation, some of the hyperparameters were set manually (for example, the learning rate was set to 0.002), while other hyperparameters and training strategies including the number of hidden layers (1, 2, 3, 4, or 5 hidden layers), the use of regularization (with and without L_1 norm), and weight initialization (random and pretraining via autoencoders) were optimized automatically.

The results showed that the best performing model was a network with three hidden layers with L_1 regularization and weights initialized via the use of an autoencoder. This model was able to classify patients and controls with an error rate of 14.2%, which was considerably lower than the error rate of the SVM model (22.3%). Importantly, the authors also reported the features (i.e., pairs of connectivity) that contributed the most to the distinction between the two groups. These included, for example, the functional connections between the thalamus and the cerebellum, between the frontal and the temporal areas, as well as between the posterior cingulate cortex and the striatum. In conclusion, the results supported the notion that DNNs allow discrimination between patients and controls with higher accuracy than "shallow" machine learning methods such as SVM and furthermore provided insight into how the choice of the hyperparameters and training strategies can influence the performance of the model.

9.3.2 Early detection in Alzheimer's disease

Neuroimaging studies typically extract a single set of features from a given source of data. However, there is evidence that providing the machine learning algorithm with the same data in different scales can improve performance. Imagine looking at the same part of an image with different resolutions. At high resolution, tiny features are captured in a small space within the image; at a lower resolution, more of the image is visible, making it easier to investigate more extensive features. The two levels of resolution, therefore, provide complementary information that can be combined to enhance the performance of a model. Based on this premise, Lu et al. (2018) proposed a novel approach for combining structural MRI (sMRI) and positron emission tomography (PET) images, each one at multiple scales, within a DNN framework. From each sMRI and PET image, three different scales (i.e., different sized patches) were extracted with 500, 1000, and 2000 voxels. The resulting total number of patches was inputted as features into a DNN. The network consisted of two parts: six independent DNNs, each corresponding to each scale of a single modality (three features sets $\times$ two imaging modalities), and another DNN to fuse the features extracted from the six DNNs. The network was pretrained using the autoencoder and subsequently fine-tuned through backpropagation with the Adam algorithm (an alternative to GD). To avoid overfitting, the authors used early stopping to stop optimization if the validation accuracy had ceased to increase for 50 epochs and dropout with a 50% rate.

Two binary classifications were carried out: (i) mild cognitive impairment (MCI) patients who did (pMCI: progressive MCI) versus did not convert to AD (sMCI: stable MCI); (ii) controls who did not developed AD (sNC: stable normal controls) versus all participants with AD at follow-up (pNC: progressive normal controls, pMCI, and AD). Results reveal an accuracy, specificity, and sensitivity of 82.9%, 83.8%, and 79.7% for the first classification and 86.4%, 86.5%, and 86.3% for the second classification. Interestingly, the DNN's accuracy in the first classification was 90%, 86.6%, and 82.4% for patients 1, 2, and 3 years away from conversion, suggesting that correct prediction is more challenging when an individual is farther away from conversion. Taken collectively, the results of this study provide an interesting example of how the flexibility of DNNs can be used to make the most of multimodal and multiscale data to improve the performance of a model.

9.3.3 Early detection of autism spectrum disorders

In one of the few longitudinal studies in brain disorders using DL, Hazlett et al. (2017) applied a DNN to distinguish infants at high risk who later developed an ASD from those who did not. In this prospective study, 34 infants at high familial risk of ASD and 145 low-risk infants were assessed at 6, 12, and 24 months of age. A total of 315 features including surface area and cortical thickness of 78 cortical brain regions and intracranial volume (all at 6 and 12 months and age-corrected) as well as sex were used as input features to predict diagnosis at 24 months of age.

The last machine learning pipeline in this study comprised a two-stage process including a first step in which a DNN was used as a dimensionality reduction and a second step in which SVM was used to perform the classification task. In other words, the main use of a DNN in this study was to learn nonlinear latent representations of the input data that would then be put through an SVM for classification. For comparison, the same pipeline was also repeated using a linear dimensionality reduction method (principal component analysis (PCA); see Chapter 12) and traditional DNN (without a separate SVM classifier). Similarly to the previous studies, the DNN was also pretrained with an autoencoder and then fine-tuned. The architecture for the DNN comprised two hidden layers with 315 neurons in the input layer, 100 in the first hidden layer, 10 in the second hidden layer, and 2 neurons in the final layer. The final two features were put through the SVM for classification. Learning rate set to 0.7 and 10-fold cross-validation was used for estimating performance.

The results showed that the proposed model distinguished the two groups with 94% accuracy, 88% sensitivity, 95% specificity, 81% positive predictive value, and 97% negative predictive value, outperforming the traditional DNN by 11% and the linear alternative by 30%. Analysis of the most important features revealed that the hyperexpansion of the surface area of the bilateral superior frontal and postcentral gyri and the enlargement of the intracranial volume were top most contributing features. In conclusion, this study illustrates how DNNs can be used as a dimensionality reduction method as a viable alternative to the commonly used PCA. Besides, it suggests that extracting nonlinear, as opposed to linear, patterns in the data may allow a more accurate representation of the complex growth patterns in the developing brain.

9.4 Conclusion

DL is currently one of the most promising and popular approaches in machine learning. As a representation learning approach, DL breaks with traditional “shallow” methods by learning meaningful features from the data through a sequence of nonlinear transformations that result in increasingly more complex and abstract features. In this chapter, we introduced the fundamentals of DL by focusing on the architecture and learning process of the simplest model—the DNN. While still in its initial stages, the application of DNNs to brain disorders has shown promising results. Although the vast majority of studies have been applied to neuroimaging data, the scope of applications is likely to expand in the next few years.

Nevertheless, several obstacles will have to be addressed to take advantage of the full potential of DL. Sample size is probably one of the most obvious ones. Being a complex nonlinear algorithm, DL thrives when applied to large samples. Indeed, the birth of DL and the (still) growing popularity that quickly followed were propelled by the increasing availability of powerful computers and large amounts of data. Although Big Data is not yet commonplace in the investigation of brain disorders, there is a clear trend toward large-scale multicenter collaborations that will increase sample sizes in the coming years. In addition, at present, the successful implementation of DL requires extensive technical expertise. However, high-level libraries such as Keras (Chollet, 2015) built on easily accessible programming languages (Python and R) are making DL more approachable to nonexperts and, with the growing popularity of this approach, it is likely that this trend will continue to grow. Despite some critical challenges, therefore, we can expect DL to continue to become an increasingly popular and useful approach to brain disorders. In the next two chapters, you will learn about two additional architectures—autoencoders and convolutional neural networks—that have shown encouraging results in the neuroscientific literature.

9.5 Key points

- DL is a family of representation learning methods capable of learning complex and abstract latent features from the data.
- DNNs learn latent features through sequential layer-wise nonlinear transformations, resulting in increasing levels of complexity and abstraction.
- Deep neural networks are structured in interconnected layers comprising neurons that perform a nonlinear transformation to the incoming data from the previous layer.
- Learning is an iterative process of adjustment of the neural network's parameters that connect neurons between adjacent layers.
- Given the massive amounts of parameters to estimate, deep neural networks are particularly prone to overfitting and tend to be computationally expensive.
- DL is a sophisticated approach with important methodological challenges to overcome; however, initial evidence has shown encouraging results in brain disorders.

References

Alain, G., & Bengio, Y. (2016). *Understanding intermediate layers using linear classifier probes*. ArXiv Preprint ArXiv:1610.01644.

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends® in Machine Learning, 2*. https://doi.org/10.1561/2200000006.

Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In G. Montavon, G. Orr, & K.-R. Müller (Eds.), *Neural networks: Tricks of the trade* (pp. 437—478). Springer.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. New York: Springer.

Calhoun, V. D., & Sui, J. (2016). Multimodal fusion of brain imaging data: A key to finding the missing link(s) in complex mental illness. *Biological Psychiatry: Cognitive Neuroscience and Neuroimaging, 1*(3), 230–244. https://doi.org/10.1016/j.bpsc.2015.12.005.

Chollet, F. (2015). *Keras*.

Choromanska, A., Henaff, M., Mathieu, M., Arous, G. B., & LeCun, Y. (2015). The loss surfaces of multilayer networks. In *Artificial intelligence and statistics* (pp. 192–204).

Dauphin, Y. N., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., & Bengio, Y. (2014a). Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in neural information processing systems* (Vol. 27, pp. 2933–2941).

Deshpande, G., Wang, P., Rangaprakash, D., & Wilamowski, B. (2015). Fully connected cascade artificial neural network architecture for attention deficit hyperactivity disorder classification from functional magnetic resonance imaging data. *Cybernetics, EEE transactions on cybernetics, 45*(12), 2668–2679. https://doi.org/10.1109/TCYB.2014.2379621.

Domingos, P. (2012). A few useful things to know about machine learning. *Communications of the ACM, 55*(10), 78–87. https://doi.org/10.1145/2347736.2347755.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. Retrieved from https://www.deeplearningbook.org/.

Grün, F., Rupprecht, C., Navab, N., & Tombari, F. (2016). *A taxonomy and library for visualizing learned features in convolutional neural networks*. ArXiv Preprint ArXiv:1606.07757.

Hazlett, H. C., Gu, H., Munsell, B. C., Kim, S. H., Styner, M., Wolff, J. J., et al. (2017). Early brain development in infants at high risk for autism spectrum disorder. *Nature, 542*(7641), 348–351. https://doi.org/10.1038/nature21369.

Hinton, G. E., Osindero, S., & Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation, 18*(7), 1527–1554.

Kim, J., Calhoun, V. D., Shim, E., & Lee, J.-H. (2016). Deep neural network with weight sparsity control and pre-training extracts hierarchical features and enhances classification performance: Evidence from whole-brain resting-state functional connectivity patterns of schizophrenia. *NeuroImage, 124*, 127–146. https://doi.org/10.1016/j.neuroimage.2015.05.018.

Kingma, D. P., & Ba, J. (2014). *Adam: A method for stochastic optimization*. ArXiv Preprint ArXiv:1412.6980.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097–1105).

Le, Q. V. (2013). Building high-level features using large scale unsupervised learning. In *Acoustics, speech and signal processing (ICASSP), 2013 IEEE international conference on* (pp. 8595–8598).

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature, 521*(7553), 436–444. https://doi.org/10.1038/nature14539.

Liu, S., Liu, S., Cai, W., Pujol, S., Kikinis, R., & Feng, D. (2014). Early diagnosis of Alzheimer's disease with deep learning. In *2014 IEEE 11th international symposium on biomedical imaging (ISBI)* (pp. 1015–1018). https://doi.org/10.1109/ISBI.2014.6868045.

Lu, D., Popuri, K., Ding, G. W., Balachandar, R., Beg, M. F., & Alzheimer's Disease Neuroimaging Initiative, A. D. N. (2018). Multimodal and multiscale deep neural networks for the early diagnosis of alzheimer's disease using structural MR and FDG-PET images. *Scientific Reports, 8*(1), 5697. https://doi.org/10.1038/s41598-018-22871-z.

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics, 5*(4), 115–133.

Montavon, G., Samek, W., & Müller, K.-R. (2018). Methods for interpreting and understanding deep neural networks. *Digital Signal Processing, 73*, 1–15.

Nowlan, S. J., & Hinton, G. E. (1992). Simplifying neural networks by soft weight-sharing. *Neural Computation, 4*(4), 473–493.

Plis, S. M., Hjelm, D. R., Salakhutdinov, R., Allen, E. A., Bockholt, H. J., Long, J. D., et al. (2014). Deep learning for neuroimaging: A validation study. *Frontiers in Neuroscience, 8*, 229. https://doi.org/10.3389/fnins.2014.00229.

Prechelt, L. (1998). Automatic early stopping using cross validation: Quantifying the criteria. *Neural Networks, 11*(4), 761–767. https://doi.org/10.1016/S0893-6080(98)00010-0.

Samek, W., Binder, A., Montavon, G., Lapuschkin, S., & Müller, K.-R. (2017). Evaluating the visualization of what a deep neural network has learned. *IEEE Transactions on Neural Networks and Learning Systems, 28*(11), 2660–2673.

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks, 61*, 85–117. https://doi.org/10.1016/j.neunet.2014.09.003.

Schnack, H. G. (2017). Improving individual predictions: Machine learning approaches for detecting and attacking heterogeneity in schizophrenia (and other psychiatric diseases). *Schizophrenia Research*. https://doi.org/10.1016/j.schres.2017.10.023.

Simonyan, K., Vedaldi, A., & Zisserman, A. (2013). *Deep inside convolutional networks: Visualising image classification models and saliency maps*. ArXiv Preprint ArXiv:1312.6034.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research, 15*, 1929–1958.

Suk, H.-I., Lee, S.-W., & Shen, D. (2015). Latent feature representation with stacked autoencoder for AD/MCI diagnosis. *Brain Structure and Function, 220*(2), 841–859. https://doi.org/10.1007/s00429-013-0687-3.

Suk, H.-I., Wee, C.-Y., Lee, S.-W., & Shen, D. (2016). State-space model with deep learning for functional dynamics estimation in resting-state fMRI. *NeuroImage, 129*, 292–307.

Vieira, S., Pinaya, W. H. L., & Mechelli, A. (2017). Using deep learning to investigate the neuroimaging correlates of psychiatric and neurological disorders: Methods and applications. *Neuroscience and Biobehavioral Reviews, 74*, 58–75. https://doi.org/10.1016/j.neurobiorev.2017.01.002.

Wolfers, T., Buitelaar, J. K., Beckmann, C. F., Franke, B., & Marquand, A. F. (2015). From estimating activation locality to predicting disorder: a review of pattern recognition for neuroimaging-based psychiatric diagnostics. *Neuroscience & Biobehavioral Reviews, 57*, 328–349.

Yosinski, J., Clune, J., Nguyen, A., Fuchs, T., & Lipson, H. (2015). *Understanding neural networks through deep visualization*. ArXiv Preprint ArXiv:1506.06579.