Deep Learning: A Primer for Psychologists

Christopher J. Urban* and Kathleen M. Gates

L.L. Thurstone Psychometric Laboratory in the Department of Psychology and Neuroscience
University of North Carolina at Chapel Hill
Chapel Hill, NC 27599

September 5, 2019

Abstract

Deep learning has revolutionized predictive modeling in computer vision and in natural language processing but is not commonly applied to psychology data. We provide an overview of machine learning and deep learning for psychologists who have a working knowledge of linear regression. We first help psychologists decide when to use deep learning by discussing how machine learning and deep learning may be used to answer both prediction-focused and causal-focused research questions. We then provide an overview of machine learning and present a general recipe for building machine learning algorithms. We define deep learning algorithms as a subset of machine learning algorithms that can extract successively more and more abstract representations of a data set by using many layers of processing. Finally, we present three basic deep learning models that generalize linear regression: The feedforward neural network, the convolutional neural network, and the recurrent neural network. We describe how these models may be applied to answer prediction-focused research questions using common data types collected by psychologists.

Keywords: Psychology, deep learning, artificial neural networks, machine learning, predictive modeling

1 Introduction

The amount of data available to psychologists in recent years has exploded. The rise of the internet has enabled researchers to recruit large, diverse, and cheap

*Correspondence to `cjurban@live.unc.edu`.

web-based samples (e.g., Buhrmester, Kwang, & Gosling, 2011; Gosling, Vazire, Srivastava, & John, 2004) as well as to utilize the vast quantities of existing web-based behavioral data (e.g., Golder & Macy, 2011; Gosling, Augustine, Vazire, Holtzman, & Gaddis, 2011; Landers, Brusso, Cavanaugh, & Collmus, 2016; Yarkoni, 2010). Smartphones and wearable sensors give researchers unprecedented opportunities to study experiential, behavioral, and physiological processes at the individual level (e.g., Hamaker & Wichers, 2017; Miller, 2012; Trull & Ebner-Priemer, 2014). Large, publicly available data sets (e.g., Institute for Quantitative Social Science Dataverse Network, dvn.iq.harvard.edu; OpenfMRI project, www.openfmri.org) let researchers answer questions that cannot be addressed in conventional lab-based settings (Yarkoni, 2012).

Statistics and artificial intelligence researchers have developed powerful, flexible *machine learning* algorithms that can be applied to these big data sets. Machine learning algorithms are typically used to predict unobserved data while trying to avoid *overfitting*, or finding patterns in a data set that do not generalize to other data sets (Hastie, Tibshirani, & Friedman, 2009). Psychologists and other social scientists have employed machine learning algorithms in diverse applications including:

- Making predictions associated with the diagnosis, prognosis, and treatment of mental illness in clinical psychology and psychiatry (e.g., Dwyer, Falkai, & Koutsouleris, 2018),
- predicting the societal impact of setting government policies in economics (Kleinberg, Ludwig, Mullainathan, & Obermeyer, 2015),
- forecasting prison violence in sociology (Baćak & Kennedy, 2018), and
- diagnostic classification of individuals using functional magnetic resonance imaging (fMRI) data in neuroscience (e.g., Pereira, Mitchell, & Botvinick, 2009).

This primer focuses on methods for prediction as well as on how predictive methods may complement other methods commonly used in psychology. Standard approaches to studying psychology primarily focus on identifying the causal mechanisms governing psychological phenomena (e.g., mediation analysis; tightly controlled experimental design), placing increased emphasis on explaining and decreased emphasis on accurately predicting some construct or outcome of interest. While explanatory methods provide many useful insights and explicitly allow for hypothesis testing, accurately predicting behaviors, diagnoses, outcomes, or emotions can also advance psychological science. Ironically, explanatory models that evidence excellent fit may not necessarily accurately predict the outcomes of interest (Wu, Harris, & Mcauley, 2007), largely because of overfitting (Yarkoni & Westfall, 2017). Research efforts that favor prediction over explanation can be highly useful. For instance, prediction-focused machine learning methods have been used to appropriately diagnose individuals (Bzdok, Engemann, Grisel, Varoquaux, & Thirion, 2018) and detect risk for future suicide attempts (Walsh, Ribeiro, & Franklin, 2017). In this way, focusing on prediction can help inform

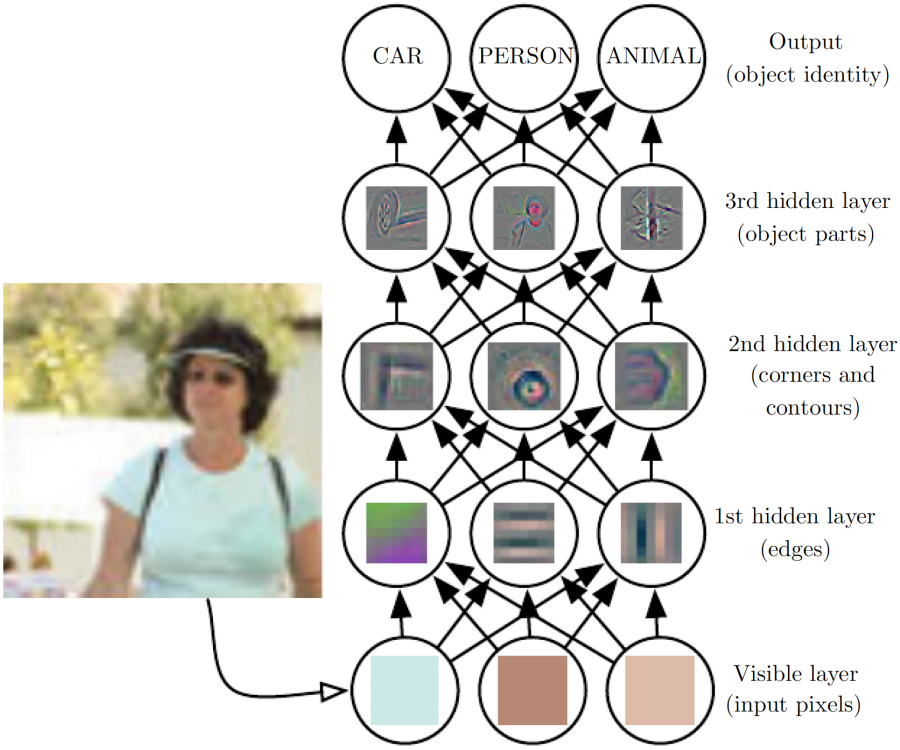
intervention, prevention, and treatment endeavors in real-life scenarios. Additionally, prediction-focused machine learning approaches can aid in the identification of important variables and relationships to include in explanatory models.

This primer introduces concepts integral for understanding *deep learning*, a machine learning paradigm that can be applied to the problems of prediction, forecasting, and classification. Deep learning is the subfield of machine learning concerned with extracting information from data in hierarchies of concepts where complicated concepts are built out of simpler concepts. We visualize this information extraction process in Figure 1 (Goodfellow, Bengio, & Courville, 2016), which contains a popular deep learning model schematic where each row of circles represents a model *layer* and arrows represent information flow through the model. The pictured model can classify objects in images. Each individual image pixel first enters the model's *visible layer*. The model's *hidden layers* identify successively more and more complicated concepts, like edges or parts of objects. Finally, the model predicts the identity of the object in the image. Psychologists often need to carefully select meaningful variables to include in machine learning models to get accurate predictions (e.g., Jiang et al., 2018). Deep learning models may avoid this pre-processing step by automatically extracting their own (though usually not directly interpretable) abstract representations of the data to use for making predictions.

Each layer in a deep learning model is just a simpler machine learning model. The most popular foundational models used to build deep learning models are called *artificial neural networks* (ANNs; LeCun, Bengio, & Hinton, 2015). In fact, ANNs and deep learning are so interlinked that some machine learning researchers consider ANNs to be another name for deep learning (e.g., Goodfellow et al., 2016). ANNs were initially inspired by biological neural mechanisms (McClelland, Rumelhart, & PDP Research Group, 1986; McCulloch & Pitts, 1943; Rosenblatt, 1958) and aim to consolidate and transfer information much like in biological learning. Specifically, the ANN modeling framework mimics animal (and human) neural processes in which neurons transfer information via synapses in a feedforward fashion. From this perspective, ANNs may be thought of as systems of nodes and edges in which each node (usually considered an "artificial neuron") produces an output that is used as input to another node. This output-input relationship is depicted graphically with an arrow (i.e., a directed edge) pointing from the output node to the input node. The nodes and edges between subsequent layers in Figure 1 follow this standard ANN representation. The earliest ANNs only had one hidden layer, but deep learning implementations of ANNs have multiple hidden layers. Deep ANNs have produced major breakthroughs in computer vision and natural language processing (LeCun et al., 2015) as well as in accurate time series forecasting (e.g., Gamboa, 2017; Karlsson, Loutfi, & Längkvist, 2014). Indeed, nearly all successful deep learning approaches have ANNs at their core.

Computer scientists have used deep ANNs to accurately predict psychological constructs of interest at unprecedented rates. For example, Suhara, Xu, and Pentland (2017), Mikelsons, Smith, Mehrotra, and Musolesi (2017), and Taylor, Jaques, Nosakhare, Sano, and Picard (2017) applied ANNs to forecast individuals'

Figure 1: Illustration of a deep learning model for classifying objects in images. Reprinted from *Deep Learning* (p. 6.), by I. Goodfellow, Y. Bengio, and A. Courville, 2016, Cambridge, MA: MIT Press. Copyright 2016 by Massachusetts Institute of Technology. Reprinted with permission.



future moods using daily diary, wearable sensor, smartphone log, and weather data. Huang, Cao, Yu, Wang, and Leow (2018) combined two kinds of ANNs to predict bipolar individuals' mood disturbances using keystroke and circadian rhythm data. Aghaei, Dimiccoli, Canton Ferrer, and Radeva (2018) asked participants to wear cameras, then used ANNs on the resulting image data to classify individuals' social interactions. In each case, ANNs outperformed the best statistical methods available for prediction, such as logistic regression and support vector machines.

Despite their state-of-the-art prediction accuracy, deep learning approaches have not been widely adopted for predictive modeling in psychology. (One notable exception is in neuroscience, where ANNs are often used to identify brain-based disorders using brain imaging data; see Vieira, Pinaya, and Mechelli (2017) for a review.) The benefits that deep learning approaches might confer to psychology beyond those offered by simpler machine learning models are therefore undetermined. Some barriers to applying deep learning for predictive modeling in psychology might include:

1. A lack of clarity about common terms like *deep learning* and fundamental concepts such as *artificial neural networks* (i.e., a *knowledge* barrier);

2. the complicated nature of building deep learning models in practice (i.e., a *complexity* barrier); and

3. a lack of standard, easy-to-use deep learning software (i.e., an *implementation* barrier).

Due to space concerns, we address the knowledge barrier in this primer and intend to address the complexity and implementation barriers in future work.

1.1 Objective

The objective of this primer is to familiarize psychologists with deep learning concepts so that they can be better prepared to learn these methods (which are largely developed in different disciplines) and to be critical consumers of articles in the psychological sciences that use deep learning methods. This primer aims to serve as an accessible reference for psychologists as they increasingly become exposed to studies that use deep learning methods. We attack this goal via the two-pronged approach described below.

First, we help psychologists determine when they should use deep learning. Common research questions that can be addressed using machine learning are presented. Cases in which deep learning models might offer benefits beyond non-deep machine learning models are explored.

Second, we de-mystify deep learning by explaining common terms using both equations and words. Machine learning algorithms are defined in a way that emphasizes their commonalities with classical statistical methods. Deep learning algorithms are defined as a particular subset of machine learning algorithms. ANN models, the most successful models under the deep learning paradigm, are

presented as a generalization of the linear regression model. Specialized ANNs that work well with image data (e.g., fMRI data) and with sequential data (e.g., daily diary data) are explained.

2 Deciding to Use Deep Learning

Deep learning promises many novel predictive modeling applications in psychology. However, it is not suitable for all, or even the majority, of applications. In this section, we aim to help psychologists determine which research questions deep learning is suited to answer. We begin by describing the kinds of research questions psychologists usually wish to answer. Next, we describe the subset of psychology research problems that can be addressed using machine learning. We then describe the smaller subset of psychology research problems on which deep learning algorithms may outperform simpler machine learning algorithms. Finally, we anticipate some future directions for applying deep learning to psychology data sets.

2.1 Common Kinds of Research Questions in Psychology

Psychologists aim to understand human behavior (Yarkoni & Westfall, 2017). Most psychologists try to achieve this goal using *statistical modeling*, which refers to the process of using mathematical tools to reach conclusions from data (Breiman, 2001). Statistical modeling allows researchers to develop and test theories by *explaining* the causal mechanisms that generated the observed data, by *predicting* new, unobserved results, and by *describing* patterns in the observed data (Shmueli, 2010). Most psychology research questions, therefore, are amenable to statistical modeling - that is, they are usually questions about the causal mechanisms giving rise to some behavioral phenomenon, about whether some behavioral phenomenon can be accurately predicted, or about how some behavioral phenomenon can be described. Due to space concerns, descriptive models are not discussed further in this primer. Currently, most psychology research questions seek to explain phenomena, so nearly all statistical models in psychology are used for causal explanation (Shmueli, 2010; Yarkoni & Westfall, 2017). Machine learning and deep learning algorithms, on the other hand, are usually thought of as statistical models for prediction (Breiman, 2001). Many psychologists may feel resistant to setting aside causal models in favor of predictive models. However, causal explanation and prediction are not at odds. In fact, explanatory modeling and predictive modeling are complementary tools that can be used separately or together to answer important psychology research questions. We discuss this point further in the following section.

2.2 Which Psychology Research Questions Are Suited to Machine Learning?

Machine learning techniques for predictive modeling can be used *directly* by themselves to answer prediction-focused research questions or *indirectly* to help answer causal explanation-focused research questions. We first discuss using machine learning techniques to directly predict outcomes of interest, which is how machine learning may offer the biggest benefits to the field of psychology (Yarkoni & Westfall, 2017).

2.2.1 Using machine learning to directly answer prediction-focused research questions

Although causal explanation is the dominant focus in most theoretical psychology research areas, prediction is often the primary goal in many applied psychology research areas such as clinical psychology, educational psychology, industrial-organizational psychology, and human factors psychology. For example, clinical psychologists are often interested in the diagnosis, prognosis, and treatment of individuals with mental illness. We can frame this interest as a prediction problem: Given some observed clinical (and maybe biological) data, can we accurately predict an individual's diagnosis, prognosis, and optimal treatment?

Even in theoretical psychology (e.g., social, cognitive, personality), many traditional research questions are most naturally addressed using predictive modeling or may be reformulated and addressed via predictive analyses. Yarkoni and Westfall (2017) describe two published examples of prediction-focused research questions in personality psychology and in cognitive psychology. First, in personality psychology, Yarkoni and Westfall note that the question of whether a person's online behaviors can be used to make inferences about their personality has often been addressed using explanatory modeling techniques (i.e., by testing for statistically significant associations between personality dimensions and online behaviors; e.g., Back et al., 2010; Gosling et al., 2011; Yarkoni, 2010). However, information about *how well* specific online behaviors predict specific personality traits is most easily obtained using predictive modeling (Kosinski, Stillwell, & Graepel, 2013). Second, in cognitive psychology, Yarkoni and Westfall describe how Rissman, Greely, and Wagner (2010) addressed the question of whether the human brain encodes traces of the true events it witnesses by using participants' fMRI activation patterns to predict whether participants had or had not previously seen images of faces. These are just two of the many examples of how theoretical psychology research questions might be addressed using predictive modeling.

The major goal of a predictive research agenda is to accurately predict outcomes of interest using new, unobserved data. Historically, prediction-focused psychology research questions like those described above have been inadequately addressed using explanatory statistical modeling techniques including p-value testing and effect size measurement, which try to make conclusions about a hypothetical population using a sample of individuals (Dwyer et al., 2018) but do

not explicitly aim to make accurate predictions with new data. Machine learning approaches to prediction improve on classical explanatory modeling approaches in two major ways. First and most importantly, unlike classical explanatory models, machine learning techniques explicitly try to avoid finding patterns in a data set that do not generalize to other data sets - that is, machine learning techniques try to avoid overfitting. *p-hacking* (Simmons, Nelson, & Simonsohn, 2011) or *data-contingent analysis* (Gelman & Loken, 2013), which refers to selecting the explanatory model that gives the best results on a particular data set, may be thought of as overfitting because the model it selects may not produce good results on new, unobserved data sets. *p*-hacking is widely thought to be a major contributor to psychology's replication crisis (Simmons et al., 2011). Machine learning techniques may therefore directly combat the replication crisis by minimizing the overfitting issues associated with *p*-hacking. Machine learning approaches to combatting overfitting are discussed in the **Learning Models that Generalize** section.

Second, machine learning algorithms often predict outcomes more accurately than classical explanatory models, especially when modeling complicated data sets. One possible reason for this is that many machine learning algorithms implicitly model large, low-order (e.g., two-way or three-way) interactions and other non-linearities that need to be explicitly specified in classical explanatory models, enabling these algorithms to obtain high predictive accuracy when the true causal relationships underlying the data are non-linear (Yarkoni & Westfall, 2017). For example, the *random forest* (Breiman, 2001) and *support vector machine* (Boser, Guyon, & Vapnik, 1992) algorithms often make accurate predictions by implicitly capturing non-linear associations between the independent and dependent variables that would need to be explicitly specified by the researcher in a linear regression model.

Another possible contributor to many machine learning algorithms' superior accuracy is their ability to leverage information from large numbers of possibly multicollinear variables to make predictions. For example, linear regression model parameters (see **Linear regression as a machine learning algorithm**) are unstable when the independent variables are multicollinear, often leading to inaccurate model predictions. Linear regression model parameters cannot even be uniquely estimated using *high-dimensional* data sets, or data sets with more variables than observations. However, extensions of linear regression called *ridge regression* (Kennard & Hoerl, 1970) and *lasso* (Tibshirani, 1996) may produce stable, accurate predictions with multicollinear and even high-dimensional data sets by emphasizing the importance of the most informative independent variables while making predictions. In sum, machine learning may benefit psychologists with prediction-focused research questions because many machine learning approaches outperform classical explanatory approaches by improving replicability and by increasing predictive accuracy, especially in complicated data sets with non-linear dependencies between a large number of variables.

2.2.2 Using machine learning to indirectly answer causal explanation-focused research questions

Machine learning techniques may be used in several ways to help build statistical models for causal explanation in psychology. The first and most straightforward way is to apply a machine learning algorithm to different sets of independent variables and to compute the algorithm's predictive accuracy on each set. Variable sets with high predictive utility for the outcome of interest may then be used to build causal models. In some cases this will be similar to the *hierarchical linear regression* (Cohen, Cohen, West, & Aiken, 2003) technique commonly used by psychologists, in which sets of independent variables are successively added to a linear regression model to determine whether the new variable sets explain a statistically significant amount of variance in the dependent variable.

Second, machine learning algorithms may be used to estimate causal effects. For example, consider a simple randomized controlled trial in which one group of patients with depression have been treated with cognitive-behavioral therapy (CBT) and another group has received no treatment. We are interested in the effects of CBT on depression symptoms. Assume that we have measured a set of independent variables for each patient. We can build one machine learning model to predict depression symptoms using the independent variables measured for patients in the control group. We can then build a separate machine learning model to predict depression symptoms for patients in the treatment group. If we apply both models to the independent variables measured for a particular patient, the difference between the outputs of these models is just the estimated causal effect of treatment for that patient. Athey (2018) provides an overview of how machine learning algorithms are being applied to solve causal inference problems.

Finally, machine learning algorithms can be used to uncover possible causal structure in a data set. For example, neuroscientists often use *Bayesian network analysis* to show the existence and direction of possible causal relationships between brain regions (Henry & Gates, 2017; Mumford & Ramsey, 2014). Additionally, automated strategies for building *structural equation models* are often used to identify possible causal relationships in psychological data (e.g., Gates & Molenaar, 2012; Marcoulides, Ing, & Hoyle, 2014).

2.3 Which Psychology Research Questions Are Suited to Deep Learning?

Deep learning is not suited for all predictive modeling applications in psychology. Deep learning algorithms often outperform simpler machine learning algorithms when most or all of the following hold (Goodfellow et al., 2016; LeCun et al., 2015):

1. There are many small, non-linear associations between variables in the data set or the variables are not easily coded in a meaningful, theoretically-relevant way;

2. a large number of observations are available; and

3. the observations are images or sequences.

We discuss each of these conditions in turn.

First, deep learning algorithms excel at discovering intricate relationships between large numbers of variables (LeCun et al., 2015). As described in the introduction, deep learning models are usually composed of many independent, modular parts called *layers* that can be stacked together in different arrangements. These layers extract progressively more and more abstract representations of the independent variables and can capitalize on many small associations between these variables to predict the outcome of interest. Many phenomena studied by psychologists are likely influenced by a large number of weak causal factors that interact in complicated ways - that is, "everything correlates with everything else" (Meehl, 1990). These phenomena may not be simple enough to be approximated by human-understandable models like linear regression (Yarkoni & Westfall, 2017). Deep learning models, which are less readily interpretable than linear regression models, may work very well in these cases (although see Montavon, Samek, and Müller (2017) for a tutorial overview of methods for interpreting deep learning models).

Additionally, although psychologists usually need to think carefully about including meaningful, theoretically-relevant variables in models to obtain accurate, generalizable predictions (called *feature engineering* in machine learning; Jiang et al., 2018), deep learning algorithms mostly avoid this pre-processing step by extracting their own (though usually not directly interpretable) representations of the independent variables to make predictions. For example, the psychological analysis of text data (Iliev, Dehghani, & Sagi, 2015), image data, and video data (Barto, Bird, Hamilton, & Fink, 2017) is usually labor-intensive, requiring human coders to look through the data and manually identify important information in each observation to use for making predictions. It might not even be clear what information will lead to the most accurate predictions. In psychological text analysis, for example, it is not always clear which features in a body of text will be the most useful for predicting outcomes like the author's mood or personality characteristics (Iliev et al., 2015). A deep learning algorithm might prove useful in this case by extracting representations of the raw text data in an automated fashion and using these representations to accurately predict mood or personality characteristics.

Second, deep learning algorithms can take advantage of large data sets to increase their predictive accuracy beyond the accuracy attainable with simpler machine learning algorithms (Goodfellow et al., 2016; LeCun et al., 2015). For example, many deep learning algorithms that performed worse than or comparably to simpler machine learning algorithms at modeling image or text data in the 1980s outperform other algorithms using the larger data sets available today (Goodfellow et al., 2016). We expect that as society becomes increasingly digitized, psychologists will have greater access to large data sets that will be difficult to effectively analyze without using deep learning. Already, psychologists can extract behavioral data sets with tens or hundreds of thousands of

observations from internet sources like social media websites, blogs, and online forums using *web scraping* (Landers et al., 2016). Cell phones and wearable sensors may produce behavioral data sets with large numbers of observations (e.g., Hamaker & Wichers, 2017). Genomics and clinical data sets are often large and may be helpful for predicting psychological outcomes (e.g., Plomin & Davis, 2009). Deep learning algorithms may be able to capitalize on these huge data sets to predict psychological outcomes with unprecedented accuracy.

Finally, deep learning algorithms often obtain higher predictive accuracy than other machine learning algorithms when the observations are images or sequences. This is because specialized deep learning models have been developed that take advantage of the structure of images and sequences to efficiently process observations. Specifically, models called *convolutional neural networks* (CNNs) have led to breakthroughs in processing image, video, and speech data, while models called *recurrent neural networks* (RNNs) have advanced predictive modeling of text, speech, and time series data (LeCun et al., 2015). CNNs and RNNs are discussed further in the **Overview of Artificial Neural Network Models** section.

In sum, deep learning may provide the most benefit to psychologists when used to predict outcomes of interest in large data sets with many correlated variables or with image or sequence observations. Although such data sets are not common in mainstream psychological research, they are readily available through web-based and clinical sources. As described in the introduction, computer scientists have already achieved promising results using deep learning to predict interesting psychological outcomes. In many cases, deep learning models outperformed simpler machine learning models, which implies that the true causal structure underlying these data sets consisted of weakly correlated interactions between large numbers of variables. We anticipate that combining many data types including survey, computer and smartphone log, social media, image and video, biometric (e.g., wearable sensor, fMRI), genomic, and Gloabl Positioning System (GPS) data will produce large data sets that deep learning models can utilize to accurately predict important psychological outcomes.

3 Machine Learning Fundamentals

Deep learning algorithms are a subset of machine learning algorithms. Understanding deep learning therefore requires some familiarity with basic machine learning concepts. In this section, we provide an overview of these foundational machine learning concepts. Readers who are familiar with machine learning basics should skip to the **Deep Learning Fundamentals** section. However, even these readers may benefit from reading **A Recipe for Building Machine Learning Algorithms**, which presents a general recipe for building machine learning algorithms.

3.1 Variable Terminology in Machine Learning

Machine learning researchers use a number of different terms to refer to different kinds of variables, which we translate for psychologists here.

In psychology and in machine learning, a *data set* usually consists of a number of variables measured for a set of objects. For example, we might measure the variables age, employment status, and height for a set of people and collect this information in a data set. A data set is typically formatted as a *data matrix* where each variable is represented by a column and each object is represented by a row. For example, a data set in which p different variables have been measured for N different objects can be formatted as follows:

$$\mathbf{X} = \overbrace{\left\{\right.}^{N \text{ observations}} \overbrace{\begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,p} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,p} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N,1} & x_{N,2} & \cdots & x_{N,p} \end{bmatrix}}^{p \text{ variables}} . \tag{1}$$

Data sets that are formatted as in equation 1 are sometimes called *tabular* data sets. In this primer, we use bold capital letters to denote matrices, bold lowercase letters to denote vectors, and non-bold, italic, lowercase letters to denote scalars such as elements of matrices or vectors. We assume that all measured values are real numbers. In equation 1, for example, we assume that every matrix element is a real number. Each row of $\mathbf{X}$, denoted $\mathbf{x}_{i,:}$, is called an *observation* or an *example*.

Both psychology and machine learning researchers often make a distinction between dependent variables and independent variables. The independent variables are usually explored to investigate their possible *statistical relationships* with the dependent variables. In psychology and machine learning, there is typically only one dependent variable, and while there can be more than one, we focus on models with one dependent variable to draw comparison with usual formulations of linear regression. Statistical relationships can be expressed in equation form as

$$\mathbf{y} = \overbrace{f(\mathbf{X})}^{\text{deterministic part}} + \overbrace{\mathbf{e}}^{\text{random part}} , \tag{2}$$

where $f(\cdot)$ is a deterministic function, $\mathbf{X}$ a $N \times p$ data matrix, $\mathbf{y}$ an $N \times 1$ vector of observations for the dependent variable, and $\mathbf{e}$ an $N \times 1$ vector of random error terms. Intuitively, equation 2 means that we are assuming that the dependent variables $\mathbf{y}$ partially depend on some independent variables we have measured (i.e., $\mathbf{X}$) and partially depend on some random information we have not measured (i.e., $\mathbf{e}$). In linear regression, for example, we assume that the dependent variables can be modeled as a weighted sum of the independent variables plus some random error. Linear regression is described in the **Linear regression as a machine learning algorithm** section.

Machine learning researchers interchangeably call independent variables *features*, *predictors*, or *input variables* and interchangeably call dependent variables

responses or *output variables*. Categorical dependent variables with two categories (e.g., "True" or "False") are called *targets*. Categorical dependent variables with more than two categories (e.g., "Red", "Green", or "Blue") are called *labels*. A categorical variable with k categories that is expressed as a vector of k dummy variables where the i^{th} dummy variable is equal to 1 and all other dummy variables are equal to 0 is called *one-hot* or *one-of-k encoded*. We write observations of one-hot variables as $[0, \ldots, 0, 1, 0, \ldots, 0]^\top$, where $(\cdot)^\top$ denotes the vector or matrix transpose. The matrix of observations of the independent variables $\mathbf{X}$ is called the *design matrix* or the *feature matrix*. The number of variables p in $\mathbf{X}$ is called the *dimensionality* of the design matrix. If the number of variables p is much larger than the number of observations N, denoted $p \gg N$, the design matrix is considered *high-dimensional*. For detailed discussions of variable terminology and dimensionality in machine learning, see Bishop (2007), Hastie et al. (2009), or Murphy (2012).

In closing this section, we note that it is not always helpful to think of a data set as a single $N \times p$ design matrix. This is true for many data sets containing potentially fascinating psychological insights, such as image or sequence data sets. For example, imagine we have a data set containing N different grayscale (i.e., "black-and-white") images. In digital form, all grayscale images are two-dimensional arrays made up of colored squares called pixels where each pixel's color is determined by a number. We can therefore think of a grayscale image as a matrix of numbers where the $(j, k)^{\text{th}}$ number describes the color of the $(j, k)^{\text{th}}$ pixel in the image. Since each observation is a matrix, our data set is no longer a single matrix - rather, it is a collection of matrices. If each image in our data set is the same shape, say $r \times c$, we can think of our data set as an $N \times r \times c$ array or *tensor* $\boldsymbol{\mathcal{X}}$ where $\mathbf{X}_{i,:,:}$ is a matrix representing the i^{th} image in the data set and $x_{i,j,k}$ is a number representing the color of the $(j, k)^{\text{th}}$ pixel in this image. In Figure 2, we visualize a data set containing four different randomly generated 4×4 grayscale images. We refrain from discussing image and sequence data sets further until the **Overview of Artificial Neural Network Models** section, where we discuss deep learning models for processing these data sets.

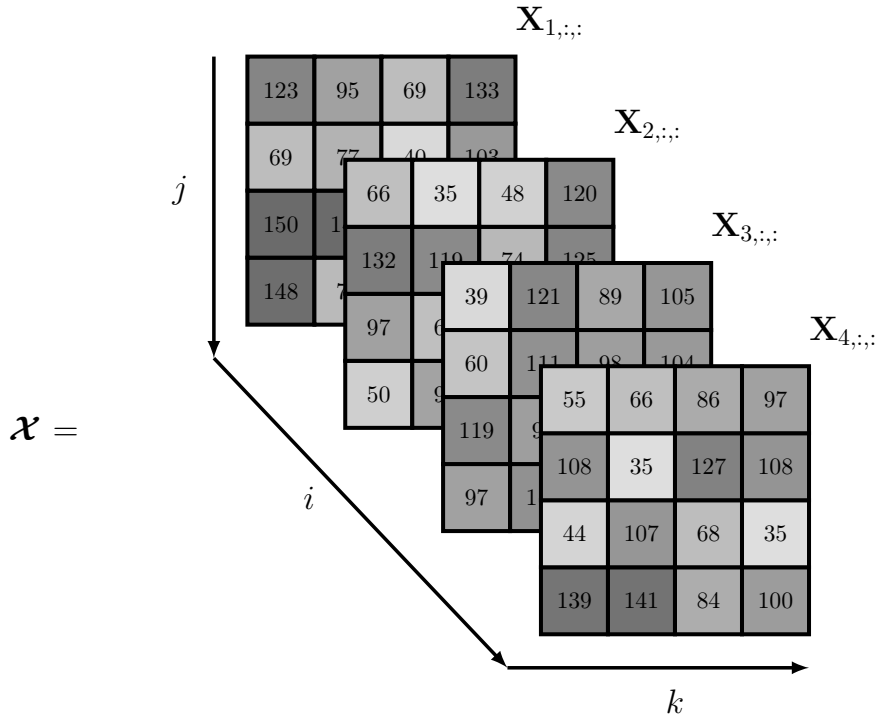
3.2 A Recipe for Building Machine Learning Algorithms

Nearly all machine learning algorithms can be put together using a recipe with four parts: A data set, a model, a cost function, and an optimization procedure (Goodfellow et al., 2016). This recipe is also used to build many classical statistical methods employed by psychologists including linear regression, factor analysis, and structural equation modeling. We discuss this recipe below.

1. **The data set.** Machine learning is concerned with extracting patterns from data. All machine learning algorithms, therefore, require at least one data set from which patterns can be extracted. Data sets are usually formatted as as in equation 1.

2. **The model.** Choosing a model means that we need to specify the deterministic part and the random part of the statistical relationship between

Figure 2: Example of an image data set containing four 4×4 random grayscale images formatted as a $4 \times 4 \times 4$ tensor.



our output variables and our input variables. Many machine learning algorithms focus only on the deterministic part and do not make assumptions about the random part. This represents a departure from typical statistical methods used by psychologists such as the *normal error linear regression model*, which makes numerous assumptions about the random error terms (Kutner, Nachtsheim, Neter, & Li, 2004). In machine learning, the model is a parameterized family of functions

$$\{\hat{f}(\cdot;\boldsymbol{\theta}) : \boldsymbol{\theta} \in \boldsymbol{\Theta}\}, \tag{3}$$

where $\boldsymbol{\theta}$ is a particular vector of parameters, $\boldsymbol{\Theta}$ is the set of all possible parameter vectors called the *parameter space*, and $\hat{f}(\cdot;\boldsymbol{\theta})$ is a function parameterized by $\boldsymbol{\theta}$ whose input is usually the data set $\mathbf{X}$. Equation 3 is read as "the set of all $\hat{f}(\cdot;\boldsymbol{\theta})$ such that $\boldsymbol{\theta}$ is an element of $\boldsymbol{\Theta}$". Intuitively, set described in equation 3 provides a possible pool of functions parameterized by $\boldsymbol{\theta}$, and the goal of the machine learning algorithm is to search the parameter space for the most appropriate parametrization such that the behavior of the function explains the observed data well - much like the goal of linear regression.

Parameters are values that affect the behavior of a function. For example, consider the simple linear regression function $\hat{f}(x;b) = bx$ with parameter vector $\boldsymbol{\theta} = [b]$. b affects the scale of the output: When b is a large number, any change in x leads to a large change in $\hat{f}(x)$; when b is near 0, any change in x leads to a small change in $\hat{f}(x)$. In equation 3, if we choose a family of functions with a fixed number of parameters (i.e., the parameter vector $\boldsymbol{\theta}$ is finite-dimensional), our model is *parametric*. If we choose a family of functions with a number of parameters that changes with the size of the data set (i.e., the parameter vector $\boldsymbol{\theta}$ is infinite-dimensional), our model is *non-parametric*.

In prediction-focused applications, also called *supervised learning*, we usually model the output variables as a function of the input variables:

$$\mathbf{y} = \hat{f}(\mathbf{X};\boldsymbol{\theta}) + \mathbf{e}. \tag{4}$$

In words, equation 4 says that $\mathbf{y}$ is modeled as a function of $\mathbf{X}$ parameterized by $\boldsymbol{\theta}$ plus a random error term $\mathbf{e}$. Supervised learning algorithms try to "learn" a good approximation $\hat{f}(\cdot;\boldsymbol{\theta})$ to the function $f(\cdot)$ in equation 2 so that the output observations $\mathbf{y}$ can be accurately predicted given the input observations $\mathbf{X}$. Accuracy for supervised learning algorithms is usually based on how "close" the predicted output values $\hat{f}(\mathbf{X};\boldsymbol{\theta})$ are to the observed output values $\mathbf{y}$. In this sense, the outputs $\mathbf{y}$ "supervise" the algorithm as it learns to approximate $f(\cdot)$. Supervised learning is called *regression* if the output variables are continuous and is called *classification* if the output variables are categorical (Bishop, 2007; Hastie et al., 2009; Murphy, 2012). Classification where the output variable has two categories is called *binary classification*. The linear regression and artificial

neural network algorithms discussed in this paper are supervised learning algorithms.

3. **The cost function.** The cost function is a method for evaluating how accurately the model is modeling the data set. Cost functions are interchangeably called *objective functions*, *loss functions*, or *error functions* (Goodfellow et al., 2016). Cost functions must take in the observed output values $\mathbf{y}$ and the predicted values $\hat{f}(\mathbf{X}; \boldsymbol{\theta})$, then output a real number. If the model is accurately modeling the data set, the cost function will output small number. If the model is not accurately modeling the data set, the cost function will output a large number. Psychology researchers might already be familiar with cost functions in the linear regression context, where we usually wish to find the line or higher-dimensional surface that best fits that observations by minimizing the squared distances between the surface and the observations. In this case, we try to minimize the mean squared error cost function, which is defined in the **Linear regression as a machine learning algorithm** section.

4. **The optimization procedure.** We often wish to make the cost function as small as possible. Minimizing the cost function ensures that our model is modeling the data set accurately. The process of minimizing the cost function is called *optimization*, *learning*, *training*, or *fitting*. During optimization, we think of the input and output values as fixed and we think of the model parameters $\boldsymbol{\theta}$ as knobs we can turn to make the cost function output smaller or larger values. In general, it is computationally infeasible (i.e., it would take a very long time) to turn these knobs manually. Instead, we use an efficient *optimization procedure* to do so for us. While we do not review optimization for artificial neural networks in this paper (see Nielsen (2015) for an accessible introduction), it might help readers to keep in mind that cost functions and optimization are used for artificial neural networks much like they are in more familiar analytic frameworks such as linear regression, and different ones can be used for any approach presented here.

3.2.1 Linear regression as a machine learning algorithm

The recipe for machine learning algorithms provided above is fairly abstract. We now present linear regression as a machine learning algorithm to make the recipe more concrete.

Linear regression starts with a data set consisting of a set of N observations of p input variables as well as a set of N observations of one output variable. We choose to model each output observation as a weighted sum of the corresponding input observations plus an intercept:

$$y_i = b_0 + b_1 x_{i,1} + b_2 x_{i,2} + \cdots + b_p x_{i,p} + e_i, \ i = 1, \ldots, N, \tag{5}$$

where y_i is the i^{th} observed value of the output variable, $x_{i,1}, x_{i,2}, \ldots, x_{i,p}$ are the i^{th} observed values of the p input variables, $b_1, b_2, \ldots, b_p$ are the weight

parameters, b_0 is the intercept, and e_i is the error or randomness in y_i that is unaccounted for by the predictors. The intercept b_0 is the predicted output value when all of the input values are equal to zero. In machine learning, intercepts are called *biases*.

It is often useful to write equation 5 concisely using matrices. To do so, we collect our output observations in an $N \times p$ matrix as in equation 1, then append a column of ones, resulting in an $N \times (p+1)$ design matrix $\mathbf{X}$. Appending a column of ones helps us include the intercept in our matrix equation. We also collect our output observations in an $N \times 1$ vector $\mathbf{y}$. Equation 5 can then be written as

$$\mathbf{y} = \mathbf{X}\mathbf{b} + \mathbf{e}, \tag{6}$$

where $\mathbf{b}$ is a $(p+1) \times 1$ vector of weight parameters including the intercept and $\mathbf{e}$ is an $N \times 1$ vector of errors. Note that equation 6 clearly models the output observations as a function of the input observations as in equation 4. Additionally, we can clearly identify the deterministic and random parts that align with equation 2.

The *mean squared error* (MSE) cost function is typically used to evaluate linear regression model accuracy:

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\big(y_i - (b_0 + b_1 x_{i,1} + b_2 x_{i,2} + \cdots + b_p x_{i,p})\big)^2. \tag{7}$$

Equation 7 is the average of the squared differences between the observed output values y_i and the predicted output values $\hat{y}_i$. MSE is a cost function because it always outputs a (positive) real number. This number should be close to zero when the observed and predicted output values are very similar and should be large when the observed and predicted output values are very different. Equation 7 is expressed in matrix form as

$$MSE = \frac{1}{N}\|\mathbf{y} - \mathbf{X}\mathbf{b}\|_2^2. \tag{8}$$

In equation 8, $\|\cdot\|_2$ denotes the ℓ^2 *norm* or the *Euclidean norm* of a vector, which computes the square root of the summation of the squared elements of the vector (i.e., for all $N \times 1$ vectors of real numbers $\mathbf{x}$, $\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{N} x_i^2}$).[1]

Finally, we optimize our linear regression model by finding the weight parameters that make our cost function as small as possible. In linear regression, this can be solved directly using an equation, but note that in many applications different parameters will need to be searched. We can achieve the lowest MSE by setting the gradient (i.e., the multi-variable derivative) of the MSE with respect

[1] The ℓ^2 norm can be thought of as the length or the size of the vector. Note that this value is squared in equation 8, thus negating the square root's impact on the value. The ℓ^2 norm is just an efficient way to write the squared sum of the differences between the observed output values and the predicted output values.

to the weight parameters to zero and solving for the weight parameters:

$$\nabla_{\mathbf{b}} \frac{1}{N} \|\mathbf{y} - \mathbf{X}\mathbf{b}\|_2^2 = 0 \implies \tag{9}$$

$$\hat{\mathbf{b}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}, \tag{10}$$

where $\nabla_{\mathbf{b}}$ denotes the gradient with respect to the weight parameters and $(\cdot)^{-1}$ denotes the matrix inverse. Psychologists may recognize equation 10, which lets us directly compute the weight parameters $\hat{\mathbf{b}}$ that minimize the MSE.

3.3 Learning Models That Generalize

The recipe described in the **A Recipe for Building Machine Learning Algorithms** section emphasizes that machine learning algorithms and many classical explanatory methods are built using the same basic ingredients. Machine learning algorithms and explanatory statistical methods mostly differ in how they justify model-building.

Explanatory statistical methods are primarily concerned with *inference* - that is, they try to make conclusions about the larger population of objects from which the data set was sampled. This is usually achieved by estimating either p-values for or confidence intervals around the model parameters $\boldsymbol{\theta}$. The overarching goal is to find the true population parameters that generated the data set and to interpret these parameters.

Machine learning algorithms are primarily concerned with *generalization* - that is, machine learning algorithms try to build models that perform well with new, previously unseen data sets (e.g., Bishop, 2007; Goodfellow et al., 2016; Hastie et al., 2009; Murphy, 2012). Although explanatory statistical methods build models intended to generalize to a population, machine learning formally tests the extent to which models generalize. This is typically achieved by dividing the data set so that some observations are placed in a *training set* and some observations are placed in a *test set*. We train our model using only the training set.[2] The final value of the cost function on the training set, called the *training error*, tells us how accurately our learning algorithm was able to model the training set. What we are really interested in, however, is the *generalization error*, which is the expected value of the cost function on a new input observation (Hastie et al., 2009). We can estimate the generalization error by using our trained model to compute the value of the cost function on the test set. We call the estimated generalization error the *test error*.

Two issues may arise after computing the training error and the test error: Underfitting or overfitting. A model suffers from *underfitting* when the training error is too large. This means that the model did not predict well even on the data used to learn the model. A model suffers from *overfitting* when the difference

[2]We split the data into a training set and a test set to mimic *direct replication*, or the process of replicating findings from one data set on a second, independent data set. Fitting the model using only the training set allows us to check whether our results replicate using the new, unseen test set data.

between the training error and the test error is too large. This indicates that the model does not generalize to data outside of the training set. We can control how likely a model is to overfit or to underfit by changing the model's *capacity*, which refers to a model's ability to approximate a wide variety of functions (Goodfellow et al., 2016). We can change a model's capacity by altering its *hypothesis space* $\mathcal{H}$, which is the set of functions a model is allowed to choose from when it is trying to approximate a particular function. We can alter a model's hypothesis space by either changing the size of its hypothesis space (i.e., changing the model's *representational capacity*) or by making it prefer certain regions of its hypothesis space (i.e., *regularizing* the model). Changing a model's representational capacity corresponds to allowing the model to choose from a different number of possible functions, while regularizing a model corresponds to influencing the specific functions the model tends to select. These concepts are demonstrated concretely in the following sections.

3.3.1 Changing a model's representational capacity

Representational capacity is an important concept in deep learning. Deep learning models like ANNs usually have very high representational capacities - that is, they can approximate a wide variety of relationships between the input and output variables. There are many ways to change the representational capacity of a deep learning model, some of which are discussed throughout the **Deep Learning Fundamentals** section. In the following paragraph, we give a simple example of changing the representational capacity of a linear regression model to demonstrate this concept.

To illustrate how altering a model's representational capacity impacts underfitting and overfitting, consider a linear regression algorithm with N observations of one input x and one output y. If we include a bias term, our model is $y_i = b_0 + b_1 x_i + e_i,\ i = 1, \ldots, N$, our parameter vector is $\boldsymbol{\theta} = [b_0, b_1]^\top$, and our cost function is $MSE = \frac{1}{N}\sum_{i=1}^{N}\big(y_i - (b_0 + b_1 x_i)\big)^2$. Our hypothesis space is $\mathcal{H} = \{f(x) = b_0 + b_1 x : b_0, b_1 \in \mathbb{R}\}$, the set of all two-dimensional lines with real-valued parameters. Our model can therefore approximate any two-dimensional line imaginable. If the data set was generated by a quadratic function like $f(x) = x^2$, however, this model will be unable to approximate the data generating function and will demonstrate underfitting (e.g., Figure 3a). We can increase our model's representational capacity by changing the model to $y_i = b_0 + b_1 x_i + b_2 x_i^2 + e_i$, which lets our model approximate any quadratic polynomial with real-valued parameters. This quadratic model will demonstrate good fit when true function underlying the data set is quadratic (e.g., Figure 3b).

In fact, we may continue adding polynomial terms to this linear regression model as long as we like. Although linear regression models with many polynomial terms are capable of approximating quadratic functions, they are also capable of approximating many other functions that fit the data set even if they were not the data-generating function (Goodfellow et al., 2016). These models are unlikely to choose the quadratic solution in practice and usually demonstrate overfitting. In Figure 3c, a linear regression model with 12 polynomial terms

demonstrates overfitting when the data-generating function is quadratic.

3.3.2 Regularizing a model

Regularization is essential when building deep learning models. ANNs and other deep learning models with high representational capacities can easily overfit a data set. Luckily, there are a large number of techniques for regularizing deep learning models to combat their tendency to overfit. We do not discuss regularization for deep learning due to space concerns, although see Goodfellow et al. (2016) for an overview. We demonstrate regularization for linear regression in the following paragraph and note that regularization for deep learning models is conceptually similar.

Consider the linear regression model with 12 polynomial terms that demonstrated overfitting in Figure 3c. To reduce overfitting, we can regularize our model by adding a penalty term called a *regularizer* to the cost function:

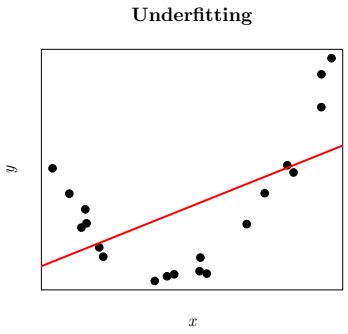
$$MSE + \overbrace{\lambda \|\boldsymbol{\theta}\|_2^2}^{\text{regularizer}}, \quad \lambda \geq 0, \tag{11}$$

where λ is a positive real number called a *regularization parameter*. Modifying a cost function by adding a regularizer that includes the ℓ^2 norm of the model parameters is called *weight decay* and is often used in deep learning algorithms. Since our optimization procedure wants to minimize this modified cost function, it will prefer parameters whose ℓ^2 norm is small. Higher values of the regularization parameter λ will make this preference stronger - that is, our optimization procedure will tend to choose smaller parameter values. In Figure 3d, regularization eliminates overfitting but reintroduces underfitting.

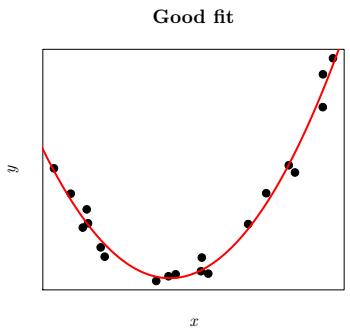
3.3.3 Hyperparameter tuning

Many machine learning algorithms have settings called *hyperparameters* that affect the behavior of the model but cannot be chosen by the algorithm itself during training. In our linear regression example, the number of polynomial terms included in our model and the value of the λ term were both hyperparameters. Our algorithm could not choose these values because increasing the polynomial degree and decreasing the value of λ would let the model approximate the observed data better and better *ad infinitum*. This, however, would result in overfitting. Instead, we choose hyperparameters by taking some observations from the training set and placing them in a *validation set*. We train many models with different hyperparameter settings on the training set, then evaluate each of their performances on the validation set. The model whose hyperparameter settings give the best performance on the validation set is evaluated on the test set. Selecting hyperparameter values this way is called *hyperparameter tuning* or *hyperparameter optimization*. Deep learning models usually have a large number of hyperparameters (e.g., the number of layers in the model; the kinds of layers used; the ways the layers are connected together). For this reason, when building deep learning models, researchers usually only test a few reasonable

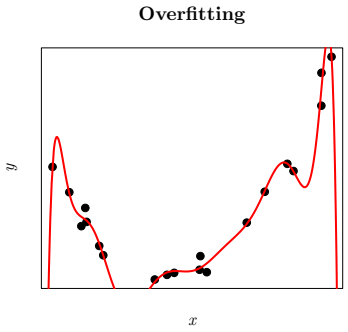
Figure 3: Illustration of the effects of altering the capacity of the linear regression algorithm on model fit.



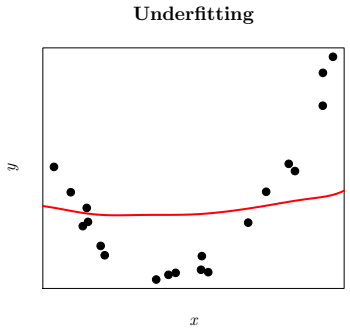
(a) Simple linear regression model.



(b) Linear regression model with a quadratic term.



(c) Linear regression model with polynomial terms up to order 12.



(d) Linear regression model with polynomial terms up to order 12 and an ℓ^2 penalty.

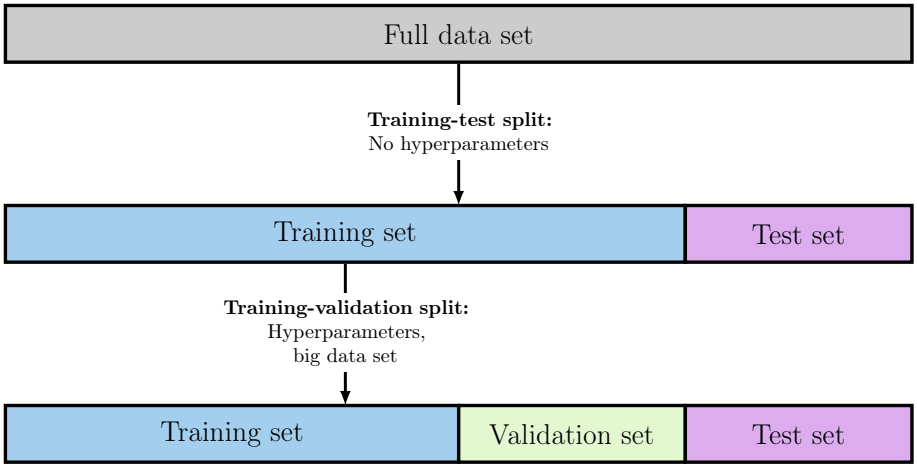


Figure 4: Schematic illustrating the validation set approach to hyperparameter tuning.

hyperparameter values until their models achieve adequate accuracy on a given data set. Additionally, the validation set approach works best with big data sets. In small data sets, a *k-fold cross-validation* approach is used for hyperparameter tuning (Yarkoni & Westfall, 2017). We illustrate hyperparameter tuning with a validation set in Figure 4.

4 Deep Learning Fundamentals

Deep learning has gotten a lot of hype in recent years (Marcus, 2018). One popular modeling approach for deep learning, artificial neural networks (ANNs), has a much longer history dating to the mid-twentieth century (McCulloch & Pitts, 1943; Rosenblatt, 1958). Both deep and non-deep ANNs have been hyped since their inception (Ripley, 2005). All of this hype has obscured what deep learning and ANNs really are. In this section, we clearly explain deep learning and ANNs for the psychologist who has a working familiarity with linear regression. We focus on three core ANN models that may be useful for predictive modeling with psychological data: Feedforward neural networks, convolutional neural networks, and recurrent neural networks.

4.1 Deep Learning Models and Terminology

What exactly makes a machine learning model a *deep learning model*? A defining feature of most deep learning models is that they map the input observations through a sequence of functions where each function in the sequence is called a *layer*. In more technical terms, nearly all deep learning models are compositions

of functions. *Function composition* is the application of one function to the output of another function. Formally, function composition is written as

$$(g \circ f)(x) = g\big(f(x)\big), \tag{12}$$

where $(g \circ f)$ is read as "g composed with f" or as "g of f". Intuitively, when we compose g with $f(x)$, we are feeding some input value x to the function f, which spits out a value $f(x)$. In turn, $f(x)$ is fed to the function g, which spits out a final value $g(f(x))$. In Figure 5a, we visualize this process with a schematic in which circles represent values and arrows represent functions.

In general, functions may be *multivariate* and *vector-valued*, which means they may take vectors as inputs and may produce vectors as outputs, respectively. Compositions of two multivariate vector-valued functions are written

$$(g \circ f)(\mathbf{x}) = g\big(f(\mathbf{x})\big) \tag{13}$$

$$= \Bigg[g_1\Big(\big[f_1(\mathbf{x}), \ldots, f_{p_1}(\mathbf{x})\big]^\top\Big), \ldots, g_{p_2}\Big(\big[f_1(\mathbf{x}), \ldots, f_{p_1}(\mathbf{x})\big]^\top\Big)\Bigg]^\top, \tag{14}$$

where $\mathbf{x}$ is a $p \times 1$ vector, $f(\mathbf{x})$ is a $p_1 \times 1$ vector, and $g\big(f(\mathbf{x})\big)$ is a $p_2 \times 1$ vector. Note that p, p_1, and p_2 (i.e., the number of elements in $\mathbf{x}$, $f(\mathbf{x})$, and $g\big(f(\mathbf{x})\big)$, respectively) are not necessarily equal. We again visualize this process with a schematic in Figure 5b. We omit function arguments in schematic diagrams for multivariate vector-valued functions to avoid clutter.

Compositions may include more than just two functions. For example, we write a composition of three multivariate vector-valued functions as

$$(g \circ f^{(2)} \circ f^{(1)})(\mathbf{x}) = g\Big(f^{(2)}\big(f^{(1)}(\mathbf{x})\big)\Big), \tag{15}$$

where $\mathbf{x}$ is a $p \times 1$ vector, $f^{(1)}(\mathbf{x})$ is a $p_1 \times 1$ vector, $f^{(2)}\big(f^{(1)}(\mathbf{x})\big)$ is a $p_2 \times 1$ vector, and $g\Big(f^{(2)}\big(f^{(1)}(\mathbf{x})\big)\Big)$ is a $p_3 \times 1$ vector. In this primer, we use parenthesized numbers in superscripts to denote ordered sequences of objects where object $(\cdot)^{(i-1)}$ comes before object $(\cdot)^{(i)}$. We may in fact compose as many functions as we wish. Compositions of $q + 1$ multivariate vector-valued functions are written

$$(g \circ f^{(q)} \circ \cdots \circ f^{(2)} \circ f^{(1)})(\mathbf{x}) = g\Bigg(f^{(q)}\Big(\ldots f^{(2)}\big(f^{(1)}(\mathbf{x})\big)\Big)\Bigg), \tag{16}$$

where $\mathbf{x}$ is a $p \times 1$ vector, $f^{(i)}\big(\ldots f^{(1)}(\mathbf{x})\big)$ is a $p_i \times 1$ vector for $i = 1, \ldots, q$, and $g\big(\ldots f^{(1)}(\mathbf{x})\big)$ is a $p_{q+1} \times 1$ vector. Nearly all deep learning models are compositions of many multivariate vector-valued functions as in equation 16, where each function has parameters we can optimize to help the model accurately predict the outcome variable.

Equation 16 helps us define important terminology for describing deep learning models. In deep learning models, each function composed together to build the

model is called a *layer*. $f^{(1)}$ is called the *first layer*, $f^{(2)}$ is called the *second layer*, and so on. $\mathbf{x}$ is called the *input layer*. Functions $f^{(1)}$ through $f^{(q)}$ are collectively called *hidden layers*. The final function g is called the *output layer*. The elements of each vector-valued layer, denoted $f_1^{(i)}, \ldots, f_{p_i}^{(i)}$ for the i^{th} layer, are called *nodes*, *artificial neurons*, or *units*. The total number of hidden layers q is called the model *depth*. Models with one hidden layer are called *shallow*. Models with more than one hidden layer are called *deep*. The number of elements in the i^{th} layer, denoted p_i, is called the *width* of layer i. The process in equation 16 is visualized with annotated layers in Figure 5c. The intuition behind choosing a model with many hidden layers is that the deep learning algorithm may decide how to use each hidden layer to best approximate the relationship between the input and output variables (Goodfellow et al., 2016).

4.2 Overview of Artificial Neural Network Models

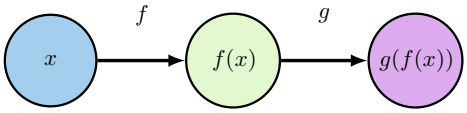
The definition of a deep learning model as a composition of functions is quite broad. In principle, we could compose together any arbitrary functions and call the resulting composition a deep learning model. However, choosing completely arbitrary functions to build deep learning models would likely produce models that are very hard to train or models that do not perform well. *Artificial neural networks* (ANNs) are one popular solution to this problem. ANNs are a broad class of non-linear statistical models that can be composed together in many layers. They have been extremely successful under the deep learning paradigm because they can be efficiently trained (using the back-propagation algorithm; Werbos, 1974) and because of their potentially high predictive accuracy.

In the following sections, we present some ANNs that may be useful for predictive modeling in psychology. We first describe feedforward neural networks (FNNs; see **Single-layer feedforward neural networks** and **Deep feedforward neural networks**). Like linear regression, FNNs are useful for predicting outcomes of interest using tabular data sets (i.e., the standard $N \times p$ data matrix $\mathbf{X}$ from equation 1). Next, we discuss convolutional neural networks (CNNs; see **Convolutional neural networks**), which are useful for predicting outcomes with image data (e.g., fMRI data). We note that CNNs are sometimes useful for other data types such as text data and other sequential data (Goodfellow et al., 2016), although we do not discuss these applications in this primer. Finally, we discuss recurrent neural networks (RNNs; see **Recurrent neural networks**), which work especially well for predicting outcomes of interest using sequential data (e.g., daily diary data; physiological trace data). We compare linear regression, FNNs, CNNs, and RNNs in Table 1.

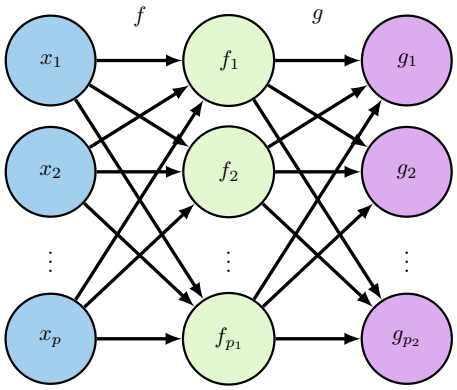
4.2.1 Single-layer feedforward neural networks

The *single-layer feedforward neural network* (FNN), also called the *single-layer perceptron*, is the simplest ANN model. The single-layer FNN only has one hidden layer and is therefore a shallow model of the form $g\big(f(\mathbf{x})\big)$ as in equation 13.

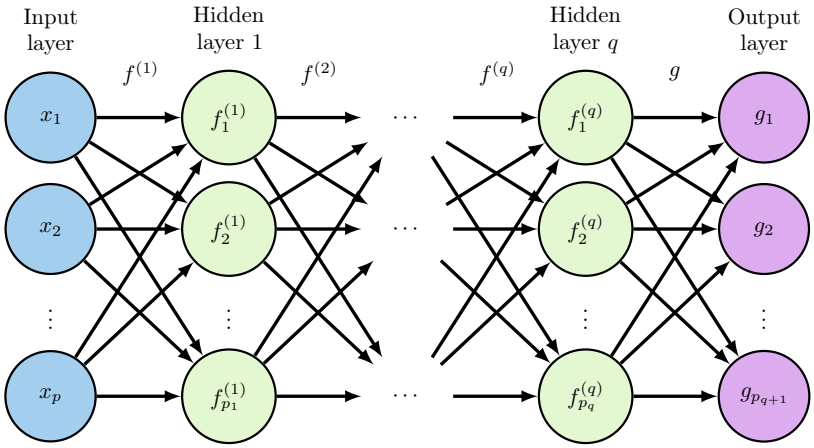
Figure 5: Schematic representations of compositions of functions.



(a) Composition of two scalar-valued functions.



(b) Composition of two multivariate vector-valued functions.



(c) Composition of $q + 1$ multivariate vector-valued functions.

	Linear regression	Feedforward neural networks	Convolutional neural networks	Recurrent neural networks
Data types	Tabular	Tabular	Images; some sequences	Sequences
Works in high-dim.[a] setting	No	Yes	Yes	Yes
Predicts continuous outcomes	Yes	Yes	Yes	Yes
Predicts categorical[b] outcomes	No	Yes	Yes	Yes

[a]High-dimensional (e.g., for tabular data, $p \gg N$).
[b]Includes binary outcomes.

Table 1: A comparison of typical use cases for the linear regression model and three basic artificial neural network models.

It is a supervised learning model that can be used either for regression or for classification.

Just like in linear regression, the single-layer FNN starts with N observations of p input variables collected in an $N \times p$ design matrix $\mathbf{X}$ as well as N observations of one output variable collected in an $N \times 1$ vector $\mathbf{y}$. To simplify notation, we drop all i subscripts indicating that we are working with the i^{th} input from $\mathbf{X}$ and the i^{th} output from $\mathbf{y}$. We now write $\mathbf{x}_{i,:}$ as $\mathbf{x}$ and write each element of $\mathbf{x}$ as x_j for $j = 1, \ldots, p$. y_i is simply written as y. This simplification is justified because single-layer FNNs and other ANNs are usually trained by feeding one randomly selected observation to the model at a time. The particular observation we choose has no impact on how the model is specified.

The single-layer FNN aims to to produce a $p_1 \times 1$ hidden layer representation $\mathbf{h}$ of each $p \times 1$ input observation $\mathbf{x}$ that can subsequently be used to predict the corresponding output observation y. To produce this hidden layer representation, the single-layer FNN first computes p_1 different weighted sums of the input values x_j plus an intercept term:

$$s_k = b_{k,0}^{\text{hx}} + \sum_{j=1}^{p} b_{k,j}^{\text{hx}} x_j, \; k = 1, \ldots, p_1, \tag{17}$$

where p_1 is the number of hidden layer nodes (as defined above). Equation 17 is simply the linear regression model in equation 5 repeated once for each hidden layer node (i.e., p_1 times) with different weight parameters for each node. Note

that the weight parameters have $(\cdot)^{\text{hx}}$ superscripts to indicate that multiplying by these weights gets us "to" the hidden layer $\mathbf{h}$ "from" the input layer $\mathbf{x}$. We use this "to-from" notation throughout this primer to unambiguously specify the weight parameters associated with each ANN layer, which is especially helpful when describing ANNs with many layers (Lipton, Berkowitz, & Elkan, 2015).

Next, the single-layer FNN applies a non-linear function $\sigma(\cdot)$ to each sum to compute the *hidden layer* or the *derived predictor* nodes:

$$h_k = \sigma(s_k),\ k = 1, \ldots, p_1. \tag{18}$$

These hidden layer nodes are elements of the $p_1 \times 1$ hidden layer representation $\mathbf{h}$, which corresponds to $f(\mathbf{x})$ from the $g\big(f(\mathbf{x})\big)$ model described in equation 13. The non-linear function $\sigma(\cdot)$ is called an *activation function* and enables the single-layer FNN to model outcomes that vary non-linearly with the input variables. Note that the same activation function is applied to all of the hidden layer nodes. The hidden layer activation function for single-layer FNNs was traditionally the *sigmoid* function

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \tag{19}$$

although modern single-layer FNNs mostly use the *rectified linear unit* or *ReLU* (Glorot, Bordes, & Bengio, 2011; Jarrett, Kavukcuoglu, Ranzato, & LeCun, 2009; Nair & Hinton, 2010)

$$\sigma(z) = \max(0, z). \tag{20}$$

The sigmoid activation function is visualized in Figure 6a and the ReLU activation function is visualized in Figure 6b. Activation functions are discussed further later in this section.

The single-layer FNN can now use the hidden layer representation $\mathbf{h}$ to predict the output observation y. To do so, it computes a weighted sum of the hidden layer values plus an intercept term, then applies another non-linear activation function $g(\cdot)$ to this sum:

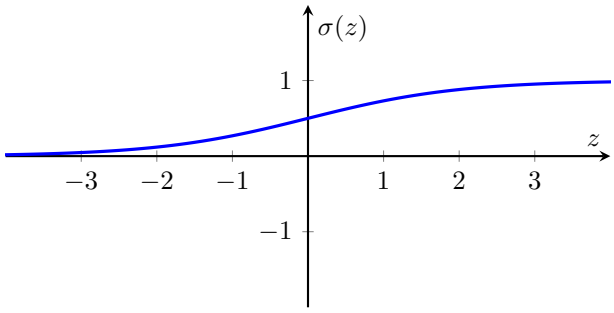
$$y = g(b_0^{\text{yh}} + \sum_{k=1}^{p_1} b_k^{\text{yh}} h_k) + e, \tag{21}$$

where $(\cdot)^{\text{yh}}$ superscripts indicate weight parameters to the output node from the hidden layer nodes and e is a random error term. The single-layer FNN is visualized in Figure 7. $\hat{y}$, the predicted output value in the final node, is typically fed to a cost function to evaluate the model's predictive accuracy. Note that intercepts are included in the schematic by multiplying the intercepts by new nodes x_0 and h_0 that are both equal to one. Psychologists may be familiar with this procedure from structural equation modeling, where it is used to include intercepts in path diagrams, often as triangles (Bollen, 1989).

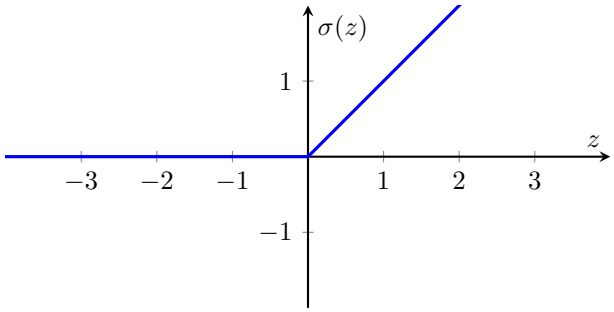
Choosing the final activation function $g(\cdot)$ determines whether our single-layer FNN will perform regression or classification. In regression, we wish to predict an output value y that may be any real number. In this case, we choose $g(\cdot)$ to be the *identity* function

$$\sigma(z) = z. \tag{22}$$

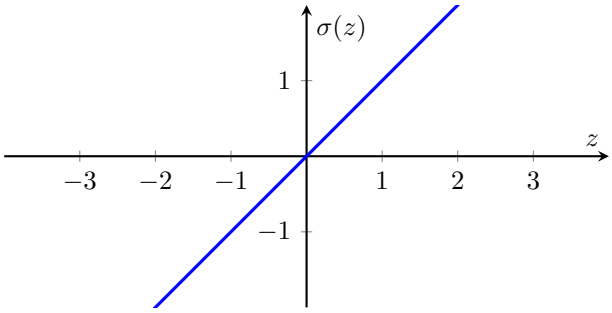
Figure 6: Common activation functions used in artificial neural network models.



(a) A graph of the sigmoid activation function.



(b) A graph of the rectified linear unit activation function.



(c) A graph of the identity activation function.

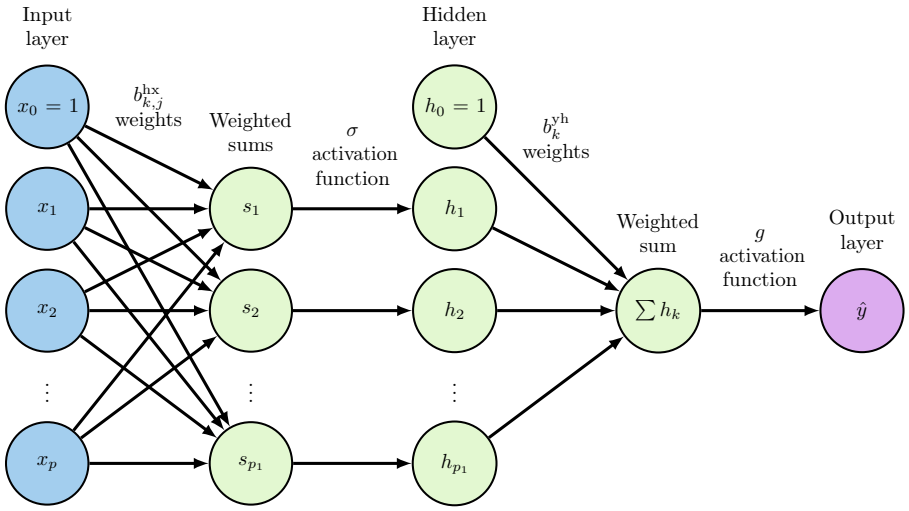


Figure 7: Schematic representation of the single-layer feedforward neural network.

Figure 6c shows that the identity function simply takes in any real number z and outputs the same real number z. In binary classification, we wish to predict an output value y that may be either zero (corresponding to the first output category) or one (corresponding to the second output category). Here, we choose $g(\cdot)$ to be the sigmoid function (equation 19). Figure 6a demonstrates that the sigmoid function takes in any real number z and outputs a number between zero and one. The number output by the sigmoid activation function represents the probability that the input observation belongs to the second output category (i.e., the category represented by $y = 1$). Finally, in classification with k categories, we wish to predict an output vector $\mathbf{y}$ that is one-hot. In this case, we choose $g(\cdot)$ to be the *softmax* function

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}, i = 1, \ldots, k, \tag{23}$$

where $\mathbf{z}$ is a $k \times 1$ vector. The softmax activation function takes in a $k \times 1$ vector of real values $\mathbf{z}$ and outputs a $k \times 1$ vector of probabilities whose i^{th} element represents the probability that the input observation belongs to category i.

Similarly to linear regression, the single-layer FNN can be expressed concisely using matrices:

$$y = g\big((\mathbf{b}^{\text{yh}})^\top \mathbf{h} + b_0^{\text{yh}}\big) + e, \tag{24}$$

$$\mathbf{h} = \sigma(\mathbf{B}^{\text{hx}}\mathbf{x} + \mathbf{b}_{\mathbf{0}}^{\text{hx}}), \tag{24a}$$

where

$$\underset{p_1 \times 1}{\mathbf{b}^{\text{yh}}} = \begin{bmatrix} b_1^{\text{yh}} \\ b_2^{\text{yh}} \\ \vdots \\ b_{p_1}^{\text{yh}} \end{bmatrix}, \quad \underset{p_1 \times 1}{\mathbf{h}} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_{p_1} \end{bmatrix}, \tag{24b–c}$$

$$\underset{p_1 \times p}{\mathbf{B}^{\text{hx}}} = \begin{bmatrix} b_{1,1}^{\text{hx}} & b_{1,2}^{\text{hx}} & \cdots & b_{1,p}^{\text{hx}} \\ b_{2,1}^{\text{hx}} & b_{2,2}^{\text{hx}} & \cdots & b_{2,p}^{\text{hx}} \\ \vdots & \vdots & \ddots & \vdots \\ b_{p_1,1}^{\text{hx}} & b_{p_1,2}^{\text{hx}} & \cdots & b_{p_1,p}^{\text{hx}} \end{bmatrix}, \underset{p \times 1}{\mathbf{x}} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix}, \underset{p_1 \times 1}{\mathbf{b}_{\mathbf{0}}^{\text{hx}}} = \begin{bmatrix} (\mathbf{b}_{\mathbf{0}}^{\text{hx}})_1 \\ (\mathbf{b}_{\mathbf{0}}^{\text{hx}})_2 \\ \vdots \\ (\mathbf{b}_{\mathbf{0}}^{\text{hx}})_{p_1} \end{bmatrix}, \tag{24d–f}$$

and the activation functions $\sigma(\cdot)$ and $g(\cdot)$ are applied to vectors element-wise. As described above, the $p_1 \times 1$ hidden layer vector $\mathbf{h}$ can be thought of as a representation of the original input observation $\mathbf{x}$ that the single-layer FNN has learned to help it predict y. $\mathbf{h}$ is in fact a vector of p_1 *latent variables*, or unobserved variables that summarize the information contained in the original observation. The concept of a latent variable may be familiar to psychologists from factor analysis and from structural equation modeling, which aim to explain the relationships between a set of observed variables in terms of a number of unobserved latent constructs.

Previously, we mentioned that ANNs can be thought of as generalizations of linear regression. We use basic algebra to show that the single-layer FNN can be reduced to the linear regression model in Appendix A, where we also briefly discuss the problem of interpreting single-layer FNNs and other ANN models.

4.2.2 Deep feedforward neural networks

Single-layer FNNs are not deep learning models because they only have one hidden layer. *Deep feedforward neural networks* (FNNs), also called *multi-layer perceptrons*, extend single-layer FNNs by including more than one hidden layer. Using many hidden layers may allow deep FNNs to model complicated, non-linear relationships between the input and output variables more efficiently than single-layer FNNs, often making deep FNNs a better modeling choice than single-layer FNNs for large, complicated data sets.

The deep FNN equations are very similar to the single-layer FNN equations. For the interested reader, we describe these equations in full detail in Appendix B. We also visualize the deep FNN in Figure 8. Notice that intercept terms are omitted from the schematic and that layer-wise summation and applying an activation function are represented using a single arrow. ANNs are typically represented this way in the deep learning literature. It is also common to omit intercept terms in path diagrams for structural equation models (Bollen, 1989).

It is not always clear when to use deep FNNs instead of single-layer FNNs in practice. In theory, both single-layer and deep FNNs can model very complicated relationships between the input and output variables. This is stated formally in

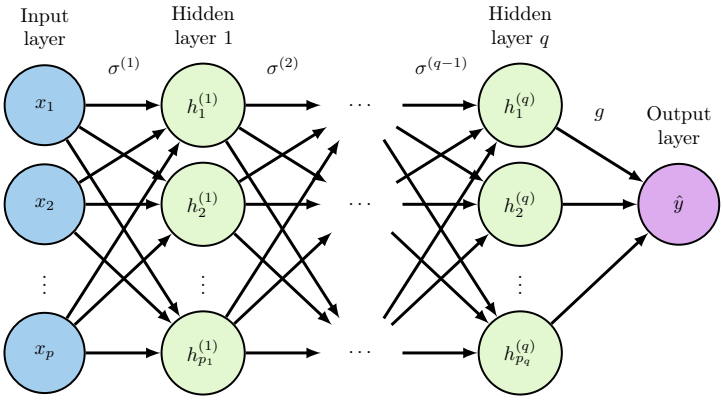


Figure 8: Schematic representation of the deep feedforward neural network.

the *universal approximation theorem*, which holds that both single-layer and deep FNNs are capable of approximating arbitrarily complicated continuous functions, given some mild assumptions[3] about the hidden layer activation functions (e.g., Csáji, 2001). This capability may be useful for problems in psychology, where the true functional relationship $f(\cdot)$ between the input $\mathbf{x}$ and the output y may be very complicated. In practice, however, single-layer FNNs may require a huge numbers of hidden layer nodes to learn the true $f(\cdot)$. Deep FNNs with many hidden layers may need fewer nodes at each layer to learn the true $f(\cdot)$ and may therefore take less time to train than single-layer FNNs (Goodfellow et al., 2016). We recommend treating the number of hidden layers as a hyperparameter: Start with one hidden layer, then increase the number of hidden layers a few times and check whether predictive accuracy improves on a validation set.

4.2.3 Convolutional neural networks

Convolutional neural networks (CNNs) are specialized ANNs for processing image data. Like FNNs, CNNs can be used for regression or classification. CNNs process images very efficiently by assuming that *local information* is important in each observation. With images, this assumption is almost always reasonable. For example, consider the image in Figure 1. Imagine that a small, square-shaped patch of pixels is randomly selected from this image. We could look at this patch and state whether or not it contains the edge of an object without needing to look at any other parts of the image - that is, only local information is important for

[3]There are a several proofs of the universal approximation theorem for both single-layer and deep FNNs. Most of these proofs assume that the hidden layer activation functions are non-constant (i.e., they are not always equal to a single, constant value), bounded (i.e., they don't shoot off to positive or negative infinity), and continuous (i.e., their graphs do not have any gaps). More recent proofs (e.g., Lu, Pu, Wang, Hu, & Wang, 2017; Sonoda & Murata, 2017) do not require the activation functions to be bounded.

identifying edges of objects. This is how CNNs work: They break up images into overlapping, squared-shaped patches of pixels, then summarize the information contained in each patch. Deep CNNs perform this summarization step at each layer, building more and more complicated concepts (e.g., parts of objects like faces or shirts) by summarizing simpler concepts (e.g., edges of objects). We only discuss single-layer CNNs in this primer, although extending single-layer CNNs to deep CNNs is straightforward and is briefly described at the end of this section.

It is helpful to think about single-layer CNNs as specialized single-layer FNNs. Recall that single-layer FNNs aim to produce a hidden layer representation $\mathbf{h}$ from each input observation $\mathbf{x}$ that can subsequently be used to predict the corresponding output observation y. Equation 24a describes how to produce $\mathbf{h}$ in a single-layer FNN: Multiply $\mathbf{x}$ by a weight matrix $\mathbf{B}^{\text{hx}}$, add an intercept vector $\mathbf{b}^{\text{hx}}$, then apply an element-wise activation function $\sigma(\cdot)$. Single-layer CNNs also aim to produce a hidden layer representation of each input image that can then be used to predict the corresponding output observation. However, single-layer CNNs replace multiplying the input image by a weight matrix $\mathbf{B}^{\text{hx}}$ with an operation called *two-dimensional discrete convolution*. Intuitively, two-dimensional discrete convolution breaks up the input image into overlapping square-shaped patches, then summarizes the information contained in each patch using a single number. We describe this process with equations in the following paragraphs.

In **Variable Terminology in Machine Learning**, we described how to think of a data set containing N grayscale images as a three-dimensional tensor $\boldsymbol{\mathcal{X}}$ (visualized in Figure 2). Recall that the i^{th} image $\mathbf{X}_{i,:,:}$ is a two-dimensional matrix of numbers, and each number determines the color of a single pixel in the image. Imagine that each image is the same shape, say $r^{\text{x}} \times c^{\text{x}}$, and that we also have an outcome value y_i associated with image i for all $i = 1, \ldots, N$. As before, we drop all i subscripts - for example, we write $\mathbf{X}_{i,:,:}$ as $\mathbf{X}$ and y_i as y.

The single-layer CNN aims to produce a hidden layer representation $\mathbf{H}$ of the input image $\mathbf{X}$ that can then be used to predict y. To produce $\mathbf{H}$, the single-layer CNN first applies two-dimensional discrete convolution to $\mathbf{X}$ to produce an intermediate representation $\mathbf{S}$. This is written in equation form as

$$\begin{gathered} s_{j,k} = (\mathbf{X} * \mathbf{B}^{\text{hx}})(j,k) = \sum_{m=j-1}^{j-r^{\text{b}}} \sum_{n=k-1}^{k-c^{\text{b}}} x_{m,n} b^{\text{hx}}_{j-m,k-n}, \\ j = 1, \ldots, r^{\text{x}} + r^{\text{b}} - 1, \; k = 1, \ldots, c^{\text{x}} + c^{\text{b}} - 1, \end{gathered} \tag{25}$$

where $*$ is called the *convolution operator*, $\mathbf{X}$ is the $r^{\text{x}} \times c^{\text{x}}$ input image, $\mathbf{B}^{\text{hx}}$ is an $r^{\text{b}} \times c^{\text{b}}$ weight matrix called the *kernel*, and $s_{j,k}$ are elements of the $(r^{\text{x}} + r^{\text{b}} - 1) \times (c^{\text{x}} + c^{\text{b}} - 1)$ matrix $\mathbf{S}$ called the *convolution* of $\mathbf{X}$ and $\mathbf{B}^{\text{hx}}$. Equation 25 looks complicated but is fairly intuitive to understand. "$\mathbf{X} * \mathbf{B}^{\text{hx}}$" is read as "$\mathbf{X}$ convolved with $\mathbf{B}^{\text{hx}}$". Convolving the input image $\mathbf{X}$ with the kernel (i.e., weight matrix) $\mathbf{B}^{\text{hx}}$ produces a new matrix $\mathbf{S}$, much like multiplying two matrices produces a new matrix. The kernel $\mathbf{B}^{\text{hx}}$ is just a small weight

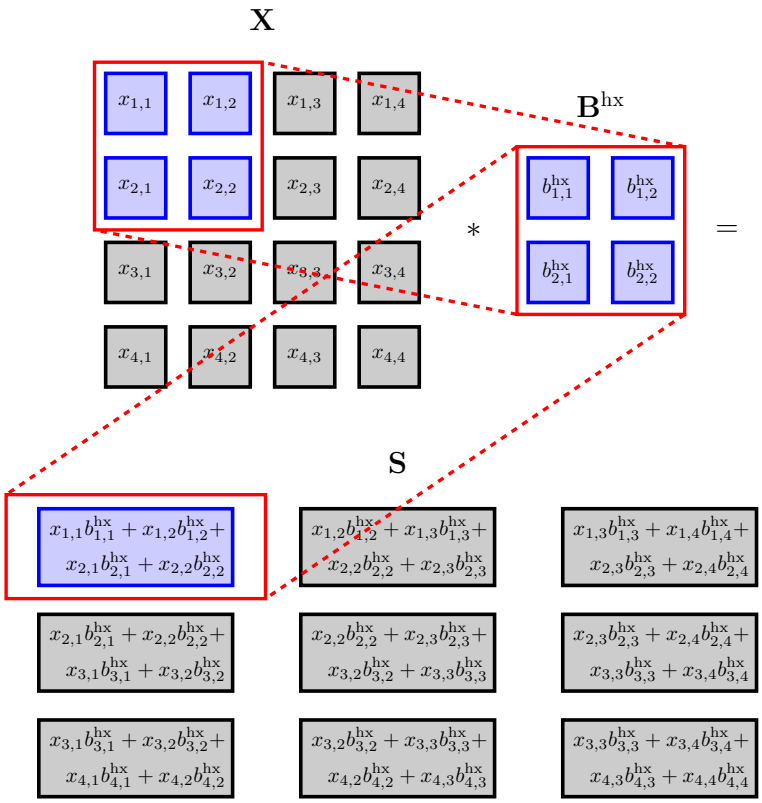


Figure 9: Example of two-dimensional discrete valid convolution with a 4×4 image input and a 2×2 kernel.

matrix that "slides" around on the input image. As it "slides", it performs element-wise multiplication with the patch of image it is currently on, then sums the result into a single output value. This "sliding" process is how single-layer CNNs use two-dimensional discrete convolution to summarize information from small, overlapping patches on the input image. We visualize two-dimensional discrete convolution with a 4×4 image and a 2×2 kernel in Figure 9. Note that our figure only visualizes the output values $s_{j,k}$ where the kernel fits completely inside the image, called the *valid convolution*.

We previously mentioned that single-layer CNNs replace multiplying the input image $\mathbf{X}$ by a weight matrix $\mathbf{B}^{\text{hx}}$ with convolving $\mathbf{X}$ and $\mathbf{B}^{\text{hx}}$. However, after performing convolution, single-layer CNNs do exactly what single-layer FNNs do to produce the hidden layer representation $\mathbf{H}$: They add an intercept

to each $s_{j,k}$, then apply an activation function $\sigma(\cdot)$:

$$\begin{gathered} h_{j,k} = \sigma(b_0^{\mathrm{hx}} + s_{j,k}), \\ j = 1, \ldots, r^{\mathrm{x}} + r^{\mathrm{b}} - 1,\ k = 1, \ldots, c^{\mathrm{x}} + c^{\mathrm{b}} - 1, \end{gathered} \tag{26}$$

where $h_{j,k}$ are elements of the resulting $(r^{\mathrm{x}} + r^{\mathrm{k}} - 1) \times (c^{\mathrm{x}} + c^{\mathrm{k}} - 1)$ hidden layer representation $\mathbf{H}$. As with single-layer FNNs, choosing a non-linear activation function helps single-layer CNNs model outcomes y that vary non-linearly with the input images $\mathbf{X}$.

We can now use the hidden layer $\mathbf{H}$ to predict the outcome y. Just like the single-layer FNN, the single-layer CNN models the outcome as a weighted sum of the hidden layer nodes, then applies a final activation function $g(\cdot)$:

$$y = g\big(b_0^{\mathrm{yh}} + \sum_{j=1}^{r^{\mathrm{h}} c^{\mathrm{h}}} b_j^{\mathrm{yh}} h_j\big) + e, \tag{27}$$

$$\mathbf{h} = \mathrm{vec}(\mathbf{H}), \tag{27a}$$

where $\mathrm{vec}(\cdot)$, the *vectorization* operation, converts the $r^{\mathrm{h}} \times c^{\mathrm{h}}$ hidden layer $\mathbf{H}$ into an $r^{\mathrm{h}} c^{\mathrm{h}} \times 1$ vector $\mathbf{h}$[4], h_j are elements of $\mathbf{h}$, and e is a random error term. As with FNNs, choosing the activation function $g(\cdot)$ determines whether the single-layer CNN will perform regression or classification.

Deep CNNs have revolutionized predictive modeling using image data as well as video, speech, and audio data (LeCun et al., 2015). Basic deep CNNs can be constructed from single-layer CNNs just like deep FNNs can be constructed from single-layer FNNs: Simply add many hidden layers, each feeding in to the next. Additionally, modifying CNNs to process non-image data is straightforward. However, the most successful deep CNNs used in practical applications can be very complicated (e.g., Krizhevsky, Sutskever, & Hinton, 2012). Goodfellow et al. (2016) describe some modifications that may improve CNN predictive accuracy as well as tips for building CNNs in practice.

4.2.4 Recurrent neural networks

Just like CNNs are specialized for processing images, *recurrent neural networks* (RNNs) are specialized for processing sequential data where the order of the observations is meaningful. Just like FNNs and CNNs, RNNs can be used for regression or classification.

Before discussing RNNs, we discuss how to describe a data set where each observation is a *sequence*. A sequence of length T is an ordered set of T observations of p different variables. Data sets usually contain many sequences. For example, daily diary data sets usually include N different individuals, each of whom is asked p questions each day for T_i days, $i = 1, \ldots, N$. We write

[4]The vectorization operation stacks the columns of $\mathbf{H}$ on top of each other. That is, $\mathrm{vec}(\mathbf{H}) = \mathbf{h} = [h_{1,1}, \ldots, h_{r^{\mathrm{h}},1}, h_{1,1}, \ldots, h_{r^{\mathrm{h}},2}, \ldots, h_{1,c^{\mathrm{h}}}, \ldots, h_{r^{\mathrm{h}},c^{\mathrm{h}}}]^\top$. Vectorizing $\mathbf{H}$ just makes it easier to sum over all of its elements.

length T_i sequences as $\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)}, \ldots, \mathbf{x}_i^{(T_i)}$, where $\mathbf{x}_i^{(t)}$ is the $p \times 1$ vector of values observed for individual i at time point t. Note that the lengths of the sequences in our data set may be different for each individual, but the number of variables is the same for all individuals. As usual, we drop all i subscripts from equations - for example, we write $\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)}, \ldots, \mathbf{x}_i^{(T_i)}$ as $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(T)}$.

We can now formulate the simple RNN model. Consider an input sequence $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(T)}$ and a corresponding output sequence $y^{(1)}, y^{(2)}, \ldots, y^{(T)}$. At each time point t, the simple RNN aims to learn a $p_1 \times 1$ hidden layer representation $\mathbf{h}^{(t)}$ of the $p \times 1$ input vector $\mathbf{x}^{(t)}$ that can then be used to predict the outcome $y^{(t)}$. Each hidden layer should also utilize information from all previous time steps when making predictions. Intuitively, the simple RNN is just a single-layer FNN with loops carrying information forward through time. The simple RNN begins by modeling the first output node as a function of the first input nodes just like the single-layer FNN:

$$y^{(1)} = g(b_0^{\text{yh}} + \sum_{k=1}^{p_1} b_k^{\text{yh}} h_k^{(1)}) + e^{(1)}, \tag{28}$$

$$h_k^{(1)} = \sigma(b_{k,0}^{\text{hx}} + \sum_{j=1}^{p} b_{k,j}^{\text{hx}} x_j^{(1)}),\ k = 1, \ldots, p_1, \tag{28a}$$

where $h_k^{(1)}$ is the k^{th} hidden layer node and $e^{(1)}$ is the error at time point 1. Equations 28 and 28a describe the usual single-layer FNN (equations 17, 18, and 21) applied to the input $\mathbf{x}^{(1)}$ at time point 1.

At subsequent time steps, however, the simple RNN models the current output node as a function of the current input nodes as well as the previous hidden layer nodes:

$$y^{(t)} = g(b_0^{\text{yh}} + \sum_{k=1}^{p_1} b_k^{\text{yh}} h_k^{(t)}) + e^{(t)},\ t = 2, \ldots, T, \tag{29}$$

$$h_k^{(t)} = \sigma(b_{k,0}^{\text{hx}} + \sum_{j=1}^{p} b_{k,j}^{\text{hx}} x_j^{(t)} + \sum_{k=1}^{p_1} b_k^{\text{hh}} h_k^{(t-1)}),\ k = 1, \ldots, p_1,\ t = 2, \ldots, T, \tag{29a}$$

where $(\cdot)^{\text{hh}}$ superscripts indicate *recurrent weight parameters* connecting hidden layer nodes at subsequent time steps and and $e^{(t)}$ is the error at time point t. Equations 29 and 29a are called *update equations* because they describe how to update the hidden state $\mathbf{h}^{(t)}$ and the predicted output $y^{(t)}$ at each time point t given the current input $\mathbf{x}^{(t)}$ and the previous hidden state $\mathbf{h}^{(t-1)}$. Any mathematical equation that starts with initial values and defines all future values as functions of previous values is called a *recurrence relation*. Recurrent neural networks are called "recurrent" because their update equations are a recurrence relation. Notice that the model re-uses the same weight parameters at each time step. Re-using weight parameters this way is called *parameter sharing* and allows RNNs to be trained with and applied to sequences of varying lengths.

The simple RNN update equations may be written concisely with matrices:

$$y^{(t)} = g\big((\mathbf{b}^{\text{yh}})^\top \mathbf{h}^{(t)} + b_0^{\text{yh}}\big) + e^{(t)},\ t = 1, \ldots, T, \tag{30}$$

$$\mathbf{h}^{(t)} = \sigma(\mathbf{B}^{\text{hx}}\mathbf{x}^{(t)} + \mathbf{B}^{\text{hh}}\mathbf{h}^{(t-1)} + \mathbf{b}_{\mathbf{0}}^{\text{hx}}),\ t = 1, \ldots, T, \tag{30a}$$

where $\mathbf{h}^{(t)}$ is the $p_1 \times 1$ hidden layer vector at time point t, $\mathbf{B}^{\text{hx}}$ is the $p_1 \times p$ matrix of weight parameters to the hidden layer nodes from the input nodes, $\mathbf{b}_{\mathbf{0}}^{\text{hx}}$ is a $p_1 \times 1$ intercept vector applied to the hidden layer nodes, $\mathbf{B}^{\text{hh}}$ is an $p_1 \times p_1$ matrix of recurrent weight parameters, $\mathbf{b}^{\text{yh}}$ is an $p_1 \times 1$ vector of weight parameters to the output node from the hidden layer nodes, b_0^{yh} is an intercept to the output nodes from the hidden layer nodes, and the initial hidden state nodes $\mathbf{h}^{(0)}$ are defined to be $\mathbf{0}$ (a $p_1 \times 1$ vector of zeros). Just like equations 29 and 29a, equations 30 and 30a are a recurrence relation - that is, all output values are functions of previous values.

To better understand the simple RNN, it is helpful to write the network in equation form without using recurrence. This is called *unfolding* the network. The simple RNN is unfolded as

$$\begin{aligned}
y^{(1)} &= g((\mathbf{b}^{\text{yh}})^\top \sigma(\mathbf{B}^{\text{hx}}\mathbf{x}^{(1)} + \mathbf{b}_{\mathbf{0}}^{\text{hx}}) + b_0^{\text{yh}}) + e^{(1)}, \\
y^{(2)} &= g\Big((\mathbf{b}^{\text{yh}})^\top \sigma\big(\mathbf{B}^{\text{hx}}\mathbf{x}^{(2)} + \mathbf{B}^{\text{hh}}\sigma(\mathbf{B}^{\text{hx}}\mathbf{x}^{(1)} + \mathbf{b}_{\mathbf{0}}^{\text{hx}}) + \mathbf{b}_{\mathbf{0}}^{\text{hx}}\big) + b_0^{\text{yh}}\Big) + e^{(2)}, \\
&\ \ \vdots \\
y^{(T)} &= g\Bigg((\mathbf{b}^{\text{yh}})^\top \sigma\Big(\mathbf{B}^{\text{hx}}\mathbf{x}^{(T)} + \ldots + \mathbf{B}^{\text{hh}}\sigma\big(\mathbf{B}^{\text{hx}}\mathbf{x}^{(2)} + \mathbf{B}^{\text{hh}}\sigma(\mathbf{B}^{\text{hx}}\mathbf{x}^{(1)} + \mathbf{b}_{\mathbf{0}}^{\text{hx}}) \\
&\qquad + \mathbf{b}_{\mathbf{0}}^{\text{hx}}\big) \ldots\Big) + b_0^{\text{yh}}\Bigg) + e^{(T)}.
\end{aligned} \tag{31}$$

Equations 31 are simply equation 30a substituted into equation 30 and written down explicitly for all time steps. Schematic representations of equations 30, 30a, and 31 are presented in Figure 10. The schematic on the left hand side of Figure 10 represents equations 30 and 30a as a single-layer FNN with a loop passing information from one time step to the next. The schematic on the right hand side of Figure 11 represents the unfolded simple RNN in equations 31 as T single-layer FNNs chained together. Both schematics are equivalent representations of the same simple RNN.

In general, the input and output sequence need not have the same length. If our input sequence has length one (i.e., $\mathbf{x}^{(1)}$) and our output sequence has length T (e.g., $y^{(1)}, y^{(2)}, \ldots, y^{(T)}$), our RNN has a *one-to-many architecture* (Figure 11a). An ANN's *architecture* refers to the number of nodes in the network and the ways that these nodes are connected (i.e., which nodes are connected as well as which weight structures and activation functions are used; Goodfellow et al., 2016). To understand when we might use an RNN with a one-to-many architecture, consider a daily diary study in which we collect information about each individual's moods and experiences once per day for several months.

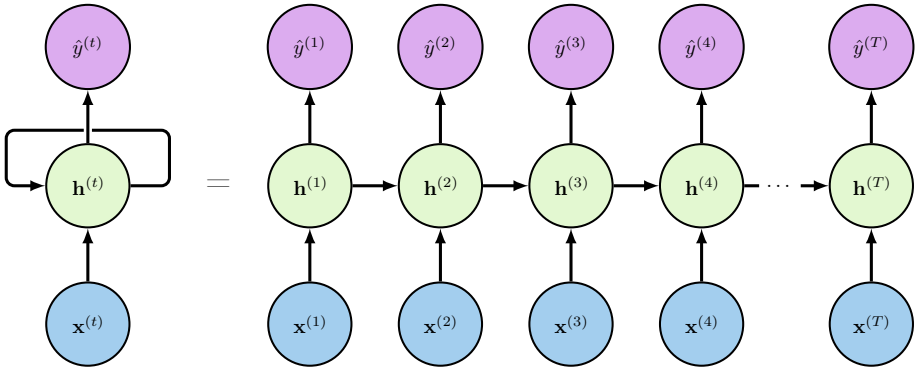
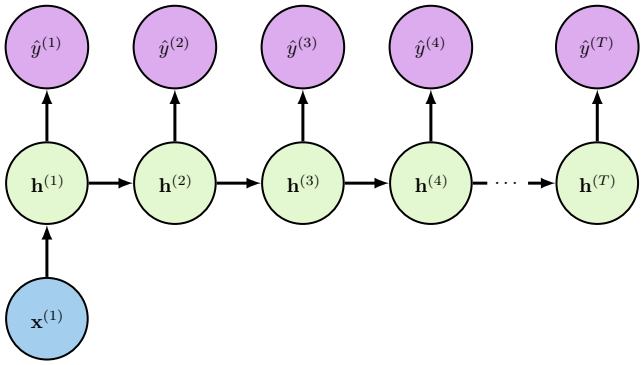


Figure 10: Schematic representation of the single-layer recurrent neural network both as a loop and as an unfolded loop.

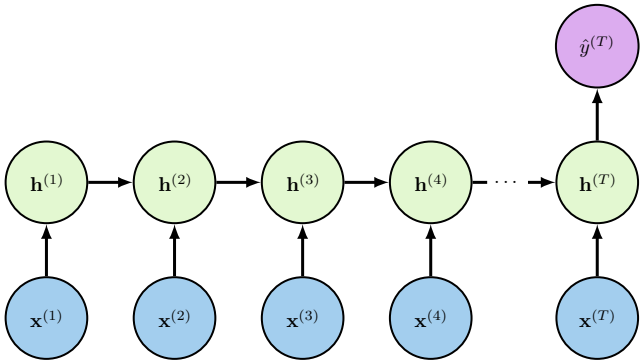
If we wish to use an individual's mood and experiences on a given day to predict their mood each day for the next three days, we would use a one-to-many architecture. If our input sequence has length T (i.e., $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(T)}$) and our output sequence has length one (i.e., $y^{(1)}$), our recurrent neural network has a *many-to-one architecture* (Figure 11b). If we wish to predict whether or not an individual will experience a depressive episode on a particular day using their past week of moods and experiences, we would use a many-to-one architecture. Finally, if both have length greater than one, our RNN has a *many-to-many architecture* (Figure 11c). If we wish to predict an individual's moods for the next three days using their past week of moods and experiences, we would use a many-to-many architecture.

RNNs are appealing in theory because they use information from the past to predict future output values. In practice, however, simple RNNs struggle to use information from very far in the past (Bengio, Simard, & Frasconi, 1994; Doya, 1993; Hochreiter, 1991; Pascanu, Mikolov, & Bengio, 2013). This may not be a problem if we only expect recent information to influence the current output value. For example, imagine collecting information about individual's moods and experiences twice per day. We might expect a person's mood early in the day to strongly influence their mood later in the day, but might expect their mood yesterday or earlier to only weakly influence their nighttime mood. A simple RNN might be well-suited to to modeling this scenario. However, if an individual experienced an extremely negative life event one month ago that we expect to strongly impact their current mood, a simple RNN might struggle to use information from so far in the past to predict a current output value. To overcome this problem with simple RNNs, specialized models called *gated recurrent neural networks* were designed to learn long-term dependencies. They are currently the most effective ANNs for modeling sequences in practice (Goodfellow et al., 2016). Lipton et al. (2015) provide an accessible overview of the most popular gated RNNs for practical applications.

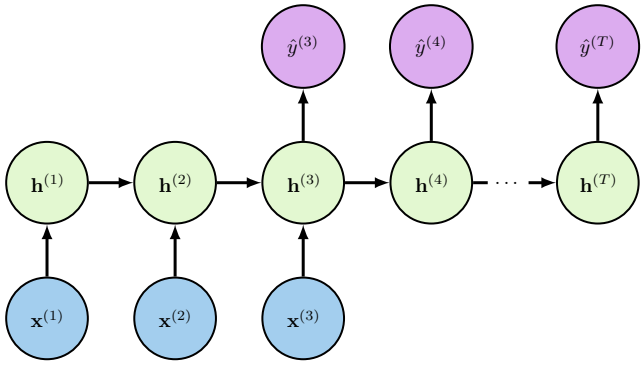
Figure 11: Schematic representations of several common recurrent neural network architectures.



(a) An example of the one-to-many recurrent neural network architecture.



(b) An example of the many-to-one recurrent neural network architecture.



(c) An example of the many-to-many recurrent neural network architecture.

5 Conclusion

Machine learning is poised to have a major impact both on predictive modeling and on causal modeling in psychology. Deep learning is a successful machine learning paradigm that has revolutionized the psychology-related fields of computer vision and natural language processing. We described how deep learning algorithms excel at modeling large data sets with small, non-linear associations between variables and with observations that are images or sequences. We anticipate that as psychologists begin to collect large data sets that combine information about participants from many sources including computers, smartphones, and wearable sensors, deep learning algorithms will outshine simpler machine learning algorithms at predicting important psychological outcomes.

In this primer, we introduced the feedforward neural network, the convolutional neural network, and the recurrent neural network as generalizations of linear regression. These models (or modifications of these models) are fundamental building blocks of advanced artificial neural networks used in large-scale scientific and industrial applications. We did not describe these state-of-the-art model architectures because deep learning research is progressing too quickly for such descriptions to be practically useful for long. Rather, we aimed to help psychologists gain some fluency in machine learning and deep learning basics. We hope that this fluency will be a first step toward enabling psychologists to draw from the machine learning literature in the same way that they have historically drawn from the statistics literature. In future work, we will discuss how to build deep learning models and will provide software implementations to help psychologists utilize deep learning to answer prediction-focused research questions.

On a final note, we wish to emphasize that machine learning and deep learning are not panaceas. In causal modeling, the *randomized controlled trial* is still a powerful experimental design for understanding the causal mechanisms that give rise to psychological phenomena (e.g., Lilienfeld, McKay, & Hollon, 2018). In predictive modeling, artificial neural networks may give worse results than simpler models like linear regression in data sets with large, linear associations between a few variables. Deep learning models that "automatically" learn complicated relationships between independent and dependent variables are not a license for psychologists to feed their data into an algorithm and hope for the best. Rather, valid psychological research requires attention to the same things it always has: Identifying which research hypotheses might be interesting and useful to investigate; translating abstract theoretical constructs into meaningful observable measurements; designing studies and collecting data sets ethically; choosing appropriate classical or modern statistical modeling techniques; and many more. Machine learning and deep learning combined with excellent research practices represent a key step toward helping psychologists accurately and reliably predict human behaviors, cognitions, and emotions.

6 Acknowledgements

We are very grateful to Van Rynald T. Liceralde and to Jeffrey A. Greene for their detailed feedback and suggestions.

References

Aghaei, M., Dimiccoli, M., Canton Ferrer, C., & Radeva, P. (2018). Towards social pattern characterization in egocentric photo-streams. *Computer Vision and Image Understanding*, *171*, 104–117. doi:10.1016/j.cviu.2018.05.001

Athey, S. (2018). The impact of machine learning on economics. (January). Retrieved from https://www.nber.org/chapters/c14009.pdf

Baćak, V. & Kennedy, E. H. (2018). Principled machine learning using the Super Learner. *Sociological Methods & Research*, (January), 1–24. doi:10.1177/0049124117747301

Back, M. D., Stopfer, J. M., Vazire, S., Gaddis, S., Schmukle, S. C., Egloff, B., & Gosling, S. D. (2010). Facebook profiles reflect actual personality, not self-idealization. *Psychological Science*, *21*(3), 372–374. doi:10.1177/0956797609360756

Baldi, P. F. & Hornik, K. (1995). Learning in linear neural networks: A survey. *IEEE Transactions on Neural Networks*, *6*(4), 837–858. doi:10.1109/72.392248

Barto, D., Bird, C. W., Hamilton, D. A., & Fink, B. C. (2017). The Simple Video Coder : A free tool for efficiently coding social video data. *Behavioral Research Methods*, *49*(4), 1563–1568. doi:10.3758/s13428-016-0787-0

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157–166.

Bishop, C. M. (2007). *Machine Learning and Pattern Recogniton*. New York, NY: Springer.

Bollen, K. A. (1989). *Structural equations with latent variables*. Wiley.

Boser, B. E., Guyon, I. M., & Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, 144–152. doi:10.1145/130385.130401

Breiman, L. (2001). Statistical Modeling : The Two Cultures. *Statistical Science*, *16*(3), 199–231.

Buhrmester, M., Kwang, T., & Gosling, S. D. (2011). Amazon's Mechanical Turk. *Perspectives on Psychological Science*, *6*(1), 3–5. doi:10.1177/1745691610393980

Bzdok, D., Engemann, D., Grisel, O., Varoquaux, G., & Thirion, B. (2018). Prediction and inference diverge in biomedicine: Simulations and real-world data. *bioRxiv*, 1–22. doi:10.1101/327437

Cohen, J., Cohen, P., West, S. G., & Aiken, L. S. (2003). *Applied multiple regression/correlation analysis for the behavioral sciences* (3rd.). New York: Psychology Press.

Csáji, B. (2001). Approximation with artificial neural networks. *MSc. thesis*, 45. doi:10.1.1.101.2647

Doya, K. (1993). Bifurcations of recurrent neural networks in gradient descent learning. *IEEE Transactions on Neural Networks*, *1*, 75–80. Retrieved from https://pdfs.semanticscholar.org/b579/27b713a6f9b73c7941f99144165396483478.pdf

Dwyer, D. B., Falkai, P., & Koutsouleris, N. (2018). Machine learning approaches for clinical psychology and psychiatry. *Annual Review of Clinical Psychology*, *14*, 91–118. doi:10.1146/annurev-clinpsy-032816-045037

Gamboa, J. (2017). Deep learning for time-series analysis. Retrieved from https://arxiv.org/pdf/1701.01887.pdf

Gates, K. M. & Molenaar, P. C. M. (2012). Group search algorithm recovers effective connectivity maps for individuals in homogeneous and heterogeneous samples. *NeuroImage*, *63*(1), 310–319. doi:10.1016/j.neuroimage.2012.06.026

Gelman, A. & Loken, E. (2013). The statistical crisis in science. *American Scientist*, *102*, 460–465. doi:10.2307/j.ctvc778jw.30

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier neural networks. *Proceedings of Machine Learning Research*, *15*, 315–323. doi:10.1.1.208.6449. arXiv: 1502.03167

Golder, S. A. & Macy, M. W. (2011). Diurnal and seasonal mood vary with work, sleep, and daylength across diverse cultures. *Science*, *333*(September), 1878–1882.

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

Gosling, S. D., Augustine, A. A., Vazire, S., Holtzman, N., & Gaddis, S. (2011). Manifestations of personality in online social networks: Self-reported Facebook-related behaviors and observable profile information. *Cyberpsychology, Behavior, and Social Networking*, *14*(9), 483–488. doi:10.1089/cyber.2010.0087

Gosling, S. D., Vazire, S., Srivastava, S., & John, O. P. (2004). Should we trust web-based studies? A comparative analysis of six preconceptions about Internet questionnaires. *American Psychologist*, *59*(2), 93–104. doi:10.1037/0003-066X.59.2.93

Hamaker, E. L. & Wichers, M. (2017). No time like the present: Discovering the hidden dynamics in intensive longitudinal data. *Current Directions in Psychological Science*, *26*(1), 10–15. doi:10.1177/0963721416666518

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: Data mining, inference, and prediction* (2nd ed.). New York: Springer. Retrieved from https://web.stanford.edu/%7B~%7Dhastie/Papers/ESLII.pdf

Henry, T. & Gates, K. (2017). Causal search procedures for fMRI: Review and suggestions. *Behaviormetrika*, *44*(1), 193–225. doi:10.1007/s41237-016-0010-8

Hochreiter, S. (1991). *Untersuchungen zu dynamischen neuronalen Netzen* (Doctoral dissertation, Technische Universität München). Retrieved from http://www.bioinf.jku.at/publications/older/3804.pdf

Huang, H., Cao, B., Yu, P. S., Wang, C. D., & Leow, A. D. (2018). DpMood: Exploiting local and periodic typing dynamics for personalized mood prediction. *Proceedings of the IEEE International Conference on Data Mining*, *2018-Novem*, 157–166. doi:10.1109/ICDM.2018.00031

Iliev, R., Dehghani, M., & Sagi, E. (2015). Automated text analysis in psychology: methods, applications, and future developments. *Language and Cognition*, *7*(2), 265–290. doi:10.1017/langcog.2014.30

Jarrett, K., Kavukcuoglu, K., Ranzato, M., & LeCun, Y. (2009). What is the best multi-stage architecture for object recognition? *Proceedings of the IEEE International Conference on Computer Vision*.

Jiang, Y., Bosch, N., Baker, R. S., Paquette, L., Ocumpaugh, J., Andres, J. M. A. L., ... Biswas, G. (2018). Expert feature-engineering vs. deep neural networks: Which is better for sensor-free affect detection? *International Conference on Artificial Intelligence in Education*, 198–211. doi:10.1007/978-3-319-93843-1_15

Karlsson, L., Loutfi, A., & Längkvist, M. (2014). A review of unsupervised feature learning and deep learning for time-series modeling. *Pattern Recognition Letters*, *42*(1), 11–24. doi:10.1016/j.patrec.2014.01.008

Kennard, R. W. & Hoerl, A. E. (1970). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, *12*(1), 55–67.

Kleinberg, J., Ludwig, J., Mullainathan, S., & Obermeyer, Z. (2015). Prediction policy problems. *American Economic Review*, *105*(5), 491–495. doi:10.1257/aer.p20151023

Kosinski, M., Stillwell, D., & Graepel, T. (2013). Private traits and attributes are predictable from digital records of human behavior. *Proceedings of the National Academy of Sciences*, *110*(15), 5802–5805. doi:10.1073/pnas.1218772110

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 1097–1105.

Kutner, M., Nachtsheim, C., Neter, J., & Li, W. (2004). *Applied linear statistical models* (5th ed.). McGraw Hill.

Landers, R. N., Brusso, R. C., Cavanaugh, K. J., & Collmus, A. B. (2016). A primer on theory-driven web scraping: Automatic extraction of big data from the Internet for use in psychological research. *Psychological Methods*, *21*(4), 475–492. doi:10.1037/met0000081

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature Methods*, *521*(1), 436–444. doi:10.1038/nmeth.3707

Lilienfeld, S. O., McKay, D., & Hollon, S. D. (2018). Why randomised controlled trials of psychological treatments are still essential. *The Lancet Psychiatry*, *5*(7), 536–538. doi:10.1016/s2215-0366(18)30045-2

Lipton, Z. C., Berkowitz, J., & Elkan, C. (2015). A critical review of recurrent neural networks for sequence learning, 1–38. Retrieved from http://arxiv.org/abs/1506.00019

Lu, Z., Pu, H., Wang, F., Hu, Z., & Wang, L. (2017). The expressive power of neural networks: A view from the width. *Conference on Neural Information Processing Systems*. Retrieved from http://arxiv.org/abs/1709.02540

Marcoulides, G. A., Ing, M., & Hoyle, R. (2014). Automated structural equation modeling strategies. In *Handbook of structural equation modeling* (Chap. 40, pp. 690–704).

Marcus, G. (2018). Deep learning: A critical appraisal, 1–24. Retrieved from https://arxiv.org/abs/1801.00631.pdf

McClelland, J. L., Rumelhart, D. E., & PDP Research Group. (1986). *Parallel Distributed Processing Vol. 1*. Cambridge, MA: MIT Press.

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

Meehl, P. E. (1990). Why summaries of research on psychological theories are often uninterpretable. *Psychological Reports*, *66*, 195–244. doi:10.4324/9780203052341

Mikelsons, G., Smith, M., Mehrotra, A., & Musolesi, M. (2017). Towards deep learning models for psychological state prediction using smartphone data: Challenges and opportunities. *Conference on Neural Information Processing Systems*. Retrieved from http://arxiv.org/abs/1711.06350

Miller, G. (2012). The smartphone psychology manifesto. *Perspectives on Psychological Science*, *7*(3), 221–237. doi:10.1177/1745691612441215

Montavon, G., Samek, W., & Müller, K. R. (2017). Methods for interpreting and understanding deep neural networks. *Digital Signal Processing: A Review Journal*, *73*, 1–15. doi:10.1016/j.dsp.2017.10.011

Mumford, J. A. & Ramsey, J. D. (2014). Bayesian networks for fMRI: A primer. *NeuroImage*, *86*, 573–582. doi:10.1016/j.neuroimage.2013.10.020

Murphy, K. P. (2012). *Machine learning: A probabilistic perspective*. Cambridge, MA: MIT Press. doi:10.1007/978-94-011-3532-0_2

Nair, V. & Hinton, G. E. (2010). Rectified linear units improve restricted Boltzmann machines. *International Conference on Machine Learnning*, (3). doi:10.1.1.165.6419

Nielsen, M. A. (2015). *Neural Networks and Deep Learning*. Determination Press. Retrieved from http://neuralnetworksanddeeplearning.com/about.html

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. Retrieved from https://arxiv.org/abs/1211.5063

Pereira, F., Mitchell, T., & Botvinick, M. (2009). Machine learning classifers and fMRI: A tutorial overview. *NeuroImage*, *45*, S199–S209. doi:10.1016/j.neuroimage.2008.11.007

Plomin, R. & Davis, O. S. P. (2009). The future of genetics in psychology and psychiatry: Microarrays, genome-wide association, and non-coding RNA. *Journal of Child Psychology and Psychiatry*, *50*(1-2), 63–71. doi:10.1038/mp.2011.182

Ripley, B. D. (2005). *Pattern recognition and neural networks*. Cambridge, UK: Cambridge University Press.

Rissman, J., Greely, H. T., & Wagner, A. D. (2010). Detecting individual memories through the neural decoding of memory states and past experience. *Proceedings of the National Academy of Sciences of the United States of America*, *107*(21), 9849–54. doi:10.1073/pnas.1001028107

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*(6), 386–408. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.335.3398%7B%5C&%7Drep=rep1%7B%5C&%7Dtype=pdf%7B%5C%%7D0Apapers://d471b97a-e92c-44c2-8562-4efc271c8c1b/Paper/p322

Shmueli, G. (2010). To explain or to predict? *Statistical Science*, *25*(3), 289–310. doi:10.2139/ssrn.1351252

Simmons, J. P., Nelson, L. D., & Simonsohn, U. (2011). False-positive psychology: Undisclosed flexibility in data collection and analysis allows presenting anything as significant. *Psychological Science*, *22*(11), 1359–1366. doi:10.1177/0956797611417632

Sonoda, S. & Murata, N. (2017). Neural network with unbounded activation functions is universal approximator. *Applied and Computational Harmonic Analysis*, *43*(2), 233–268. doi:10.1016/j.acha.2015.12.005

Suhara, Y., Xu, Y., & Pentland, A. S. (2017). Forecasting depressed mood based on self-reported histories via recurrent neural networks. *International World Wide Web Conference Committee*, 715–724.

Taylor, S., Jaques, N., Nosakhare, E., Sano, A., & Picard, R. (2017). Personalized multitask learning for predicting tomorrow's mood, stress, and health. *IEEE Transactions on Affective Computing*, (99). Retrieved from https://affect.media.mit.edu/pdfs/17.TaylorJaques-PredictingTomorrowsMoods.pdf

Tibshirani, R. (1996). Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society, Series B (Methodological)*, *58*(1), 267–288.

Trull, T. J. & Ebner-Priemer, U. (2014). The role of ambulatory assessment in psychological science. *Current Directions in Psychological Science*, *23*(6), 466–470. doi:10.1177/0963721414550706

Vieira, S., Pinaya, W. H., & 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. doi:10.1016/j.neubiorev.2017.01.002

Walsh, C. G., Ribeiro, J. D., & Franklin, J. C. (2017). Predicting risk of suicide attempts over time through machine kearning. *Clinical Psychological Science*, *5*(3), 457–469. doi:10.1177/2167702617691560

Werbos, P. J. (1974). *Beyond regression: New tools for prediction and analysis in the behavioral sciences* (Doctoral dissertation, Harvard University). Retrieved from https://ci.nii.ac.jp/naid/10004070196/

Wu, S., Harris, T. J., & Mcauley, K. B. (2007). The use of simplified or misspecified models: Linear case. *The Canadian Journal of Chemical Engineering*, *85*(4), 386–398. doi:10.1002/cjce.5450850401

Yarkoni, T. (2010). Personality in 100,000 words: A large-scale analysis of personality and word use among bloggers. *Journal of Research in Personality*, *44*(3), 363–373. doi:10.1016/j.jrp.2010.04.001

Yarkoni, T. (2012). Psychoinformatics: New horizons at the interface of the psychological and computing sciences. *Current Directions in Psychological Science*, *21*(6), 391–397. doi:10.1177/0963721412457362

Yarkoni, T. & Westfall, J. (2017). Choosing prediction over explanation in psychology: Lessons from machine learning. *Perspectives on Psychological Science*, *12*(6), 1100–1122. doi:10.1177/1745691617693393

Appendix A Single-Layer Feedforward Neural Networks Generalize Linear Regression

In this appendix, we show that linear regression is a special case of the single-layer feedforward neural network (FNN). We discuss the problem of interpreting the single-layer FNN's parameters as well as the more general problem of overparameterization in artificial neural networks (ANNs).

To show that linear regression is a special case of the single-layer FNN, we first choose the single-layer FNN's activation functions $g(\cdot)$ and $\sigma(\cdot)$ to be identity functions:

$$g(z) = z,\ \sigma(z) = z. \tag{A.1}$$

Our single-layer FNN can then be written as

$$y = b_0^{\text{yh}} + \sum_{k=1}^{p_1} b_k^{\text{yh}} h_k + e, \tag{A.2}$$

$$h_k = b_{k,0}^{\text{hx}} + \sum_{j=1}^{p} b_{k,j}^{\text{hx}} x_j,\ k = 1, \ldots, p_1. \tag{A.2a}$$

We can substitute equation A.2a into equation A.2 and rearrange to obtain

$$y = b_0^{\text{yh}} + \sum_{k=1}^{p_1} b_k^{\text{yh}} h_k + e \tag{A.2}$$

$$= b_0^{\text{yh}} + \sum_{k=1}^{p_1} b_k^{\text{yh}} (b_{k,0}^{\text{hx}} + \sum_{j=1}^{p} b_{k,j}^{\text{hx}} x_j) + e \tag{A.3}$$

$$= b'_0 + \sum_{j=1}^{p} b'_j x_j + e, \tag{A.4}$$

where

$$b'_0 = b_0^{\text{yh}} + \sum_{k=1}^{p_1} b_k^{\text{yh}} b_{k,0}^{\text{hx}},\ b'_j = \sum_{k=1}^{p_1} b_k^{\text{yh}} b_{k,j}^{\text{hx}},\ j = 1, \ldots, p. \tag{A.4a}$$

Equation A.4 clearly has the same form as the linear regression model in equation 11.[5] We have therefore demonstrated that the single-layer FNN with identity activation functions reduces to the linear regression model.

We can directly interpret the $b'_0, b'_1, \ldots, b'_p$ parameters just like we can directly interpret the $b_0, b_1, \ldots, b_p$ parameters in linear regression. However, the single-layer FNN parameters (i.e., the parameters with $(\cdot)^{\mathrm{yh}}$ and $(\cdot)^{\mathrm{hx}}$ superscripts) cannot be interpreted. This is because the single-layer FNN is *overparameterized* - that is, the model has more parameters than equations. Specifically, the single-layer FNN has $(p+2) \times p_1 + 1$ parameters total versus $p+1$ equations total (see equations 24b, 24d, and 24f to count parameters and equations A.4a to count equations). Infinitely many sets of single-layer FNN parameters will satisfy equations A.4a and produce a valid model, so any particular set of single-layer FNN parameters we choose will have no intrinsic meaning (Kutner et al., 2004).

We showed that linear regression is a special case of the single-layer FNN to help readers better understand the relationship between these models. In practice, using a single-layer FNN to perform linear regression is overly complicated. Our toy example did, however, highlight a very real, practical issue: Overparameterization. Nearly all ANNs are overparameterized and therefore have uninterpretable parameters. The uninterpretability of ANN parameters is sometimes seen as a major shortcoming. However, ANNs often achieve much higher predictive accuracy than simpler, directly interpretable models like linear regression, especially when the true causal structure underlying the data set contains weakly correlated interactions between large numbers of variables. Additionally, recent work has explored methods for interpreting ANNs that avoid the overparameterization problem (Montavon et al., 2017). These methods aim to interpret the concepts learned by the ANN's hidden layers and to identify the most important input variables used by the ANN to make predictions.

Appendix B Deep Feedforward Neural Network Equations

In this appendix, we describe the deep feedforward neural network (FNN) using equations.

To produce its first p_1 hidden layer nodes, the deep FNN computes p_1 weighted sums of the input values x_j plus an intercept, then applies an activation function $\sigma^{(1)}(\cdot)$ to each sum:

$$h_k^{(1)} = \sigma^{(1)}(b_{k,0}^{\mathrm{h_1x}} + \sum_{j=1}^{p} b_{k,j}^{\mathrm{h_1x}} x_j), \; k = 1, \ldots, p_1, \tag{B.1}$$

where $(\cdot)^{\mathrm{h_1x}}$ superscripts indicate to weight parameters to the first hidden layer nodes from the input nodes. Equation B.1 is clearly the same as equation 17

[5] When we use a suitable optimization procedure, our estimates for the $b'_0, b'_1, \ldots, b'_p$ parameters will converge to the $\hat{b}_0, \hat{b}_1, \ldots, \hat{b}_p$ parameter estimates produced by the usual linear regresssion algorithm (see equation 10; e.g., Baldi & Hornik, 1995).

substituted into equation 18 - that is, the deep FNN's first hidden layer is clearly computed the same way as the single-layer FNN's single hidden layer.

The deep FNN uses the hidden layer nodes at layer $\ell - 1$ to produce the hidden layer nodes at layer ℓ. Specifically, successive hidden layer nodes are produced by computing weighted sums over the previous hidden layer nodes plus intercept terms, then applying activation functions to these sums:

$$h_k^{(\ell)} = \sigma^{(\ell)}(b_{k,0}^{\mathrm{h}_\ell \mathrm{h}_{\ell-1}} + \sum_{j=1}^{p_{\ell-1}} b_{k,j}^{\mathrm{h}_\ell \mathrm{h}_{\ell-1}} h_j^{(\ell-1)}),\ k = 1, \ldots, p_\ell,\ \ell = 2, \ldots, q, \tag{B.2}$$

where q denotes the model depth, p_ℓ denotes the number of nodes at hidden layer ℓ, and $(\cdot)^{\mathrm{h}_\ell \mathrm{h}_{\ell-1}}$ superscripts indicate weight parameters to hidden layer ℓ nodes from hidden layer $\ell - 1$ nodes.

Finally, the output node is predicted by taking a weighted sum of the final hidden layer nodes plus an intercept term, then applying an activation function $g(\cdot)$ to this sum:

$$y = g(b_0^{\mathrm{yh_q}} + \sum_{k=1}^{p_q} b_k^{\mathrm{yh_q}} h_k^{(q)}) + e, \tag{B.3}$$

where $(\cdot)^{\mathrm{yh_q}}$ superscripts indicate weight parameters to the output node from the final hidden layer nodes and e is a random error term. As with the single-layer FNN, choice of the activation function $g(\cdot)$ determines whether the deep FNN will perform regression or classification.

Like single-layer FNNs, deep FNNs can be represented concisely using matrices:

$$y = g\big((\mathbf{b}^{\mathrm{yh_q}})^\top \mathbf{h}^{(q)} + b_0^{\mathrm{yh_q}}\big) + e, \tag{B.4}$$

$$\mathbf{h}^{(\ell)} = \sigma^{(\ell)}(\mathbf{B}^{\mathrm{h}_\ell \mathrm{h}_{\ell-1}} \mathbf{h}^{(\ell-1)} + \mathbf{b_0}^{\mathrm{h}_\ell \mathrm{h}_{\ell-1}}),\ \ell = 2, \ldots, q, \tag{B.4a}$$

$$\mathbf{h}^{(1)} = \sigma(\mathbf{B}^{\mathrm{h_1 x}} \mathbf{x} + \mathbf{b_0}^{\mathrm{h_1 x}}), \tag{B.4b}$$

where $\mathbf{x}$ is the $p \times 1$ input, $\mathbf{B}^{\mathrm{h_1 x}}$ is the $p_1 \times p$ weight matrix from the input to the first hidden layer, $\mathbf{b_0}^{\mathrm{h_1 x}}$ is the $p_1 \times 1$ intercept vector from the input to the first hidden layer, $\mathbf{h}^{(\ell)}$ is the $p_\ell \times 1$ hidden layer representation at layer ℓ, $\mathbf{B}^{\mathrm{h}_\ell \mathrm{h}_{\ell-1}}$ is the $p_\ell \times p_{\ell-1}$ weight matrix from hidden layer $\ell - 1$ to hidden layer ℓ, $\mathbf{b_0}^{\mathrm{h}_\ell \mathrm{h}_{\ell-1}}$ is the $p_\ell \times 1$ intercept vector from hidden layer $\ell - 1$ to hidden layer ℓ, $\mathbf{b}^{\mathrm{yh_q}}$ is the $p_q \times 1$ weight vector from the final hidden layer to the output, and $b_0^{\mathrm{yh_q}}$ is the intercept from the final hidden layer to the output.