AMERICAN PSYCHOLOGICAL ASSOCIATION

Psychological Methods

ISSN: 1082-989X

2021, Vol. 26, No. 6, 743–773
https://doi.org/10.1037/met0000374

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

Abstract

Deep learning has revolutionized predictive modeling in topics such as computer vision and natural language processing but is not commonly applied to psychological data. In an effort to bring the benefits of deep learning to psychologists, we provide an overview of deep learning for researchers who have a working knowledge of linear regression. We first discuss several benefits of the deep learning approach to predictive modeling. We then present three basic deep learning models that generalize linear regression: the feedforward neural network (FNN), the recurrent neural network (RNN), and the convolutional neural network (CNN). We include concrete toy examples with R code to demonstrate how each model may be applied to answer prediction-focused research questions using common data types collected by psychologists.

Translational Abstract

Deep learning has been successfully used to solve complex problems in computer vision and in natural language processing but is rarely used in psychology. In this primer, we provide an overview of deep learning in an effort to bring the benefits of deep learning to psychologists. We first discuss several benefits of using deep learning algorithms to predict important outcomes. We then present three basic deep learning models: the feedforward neural network (FNN), the recurrent neural network (RNN), and the convolutional neural network (CNN). We use toy examples with R code to demonstrate how these models may be applied to predict important outcomes using the kinds of data sets typically collected by psychologists.

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

Supplemental materials: https://doi.org/10.1037/met0000374.supp

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 web-based samples (e.g., Buhrmester et al., 2011; Gosling et al., 2004) as well as to utilize the vast quantities of existing web-based behavioral data (e.g., Golder & Macy, 2011; Gosling et al., 2011; Landers et al., 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. Oftentimes these algorithms are applied to predict some outcome, such as to classify an individual into some category based on the data or to predict a future event. In these cases, machine learning algorithms are typically used in conjunction with techniques to prevent *overfitting*, or finding spurious patterns in a data set that do not generalize to other data sets (Hastie et al., 2009). Researchers have employed machine learning algorithms to answer a variety of prediction-focused psychology research questions using the kinds of data sets described above. Social media data have been leveraged to identify important predictors of mental health issues including depression (Schwartz et al., 2014), posttraumatic stress disorder (Coppersmith et al., 2014), suicidal ideation (Coppersmith et al., 2016; De Choudhury et al., 2016), and schizophrenia (Mitchell et al., 2015). Analyses of smartphone usage and wearable sensor data have pinpointed predictors of current and future cognitive states such as moods and emotions (e.g., LiKamWa, 2012; Mehrotra et al., 2017; Rachuri et al., 2010) with a particular focus on depressive states (e.g., Canzian & Musolesi, 2015; Mehrotra et al., 2016; Saeb et al., 2015). Large, publicly available clinical data sets have been used to make

This article was published Online First April 1, 2021.

Christopher J. Urban https://orcid.org/0000-0003-0768-767X

This material is based on work supported by the National Science Foundation Graduate Research Fellowship under Grant DGE-1650116. We are grateful to Yifeng Shi, Van Rynald T. Liceralde, and Jeffrey A. Greene for their helpful feedback and support.

Correspondence concerning this article should be addressed to Christopher J. Urban, L. L. Thurstone Psychometric Laboratory in the Department of Psychology and Neuroscience, University of North Carolina at Chapel Hill, Campus Box #3270, Davie Hall, Chapel Hill, NC 27599-3270, United States. Email: cjurban@live.unc.edu

predictions associated with the diagnosis, prognosis, and treatment of mental illness in clinical psychology, psychiatry, and neuroscience (e.g., Dwyer et al., 2018).

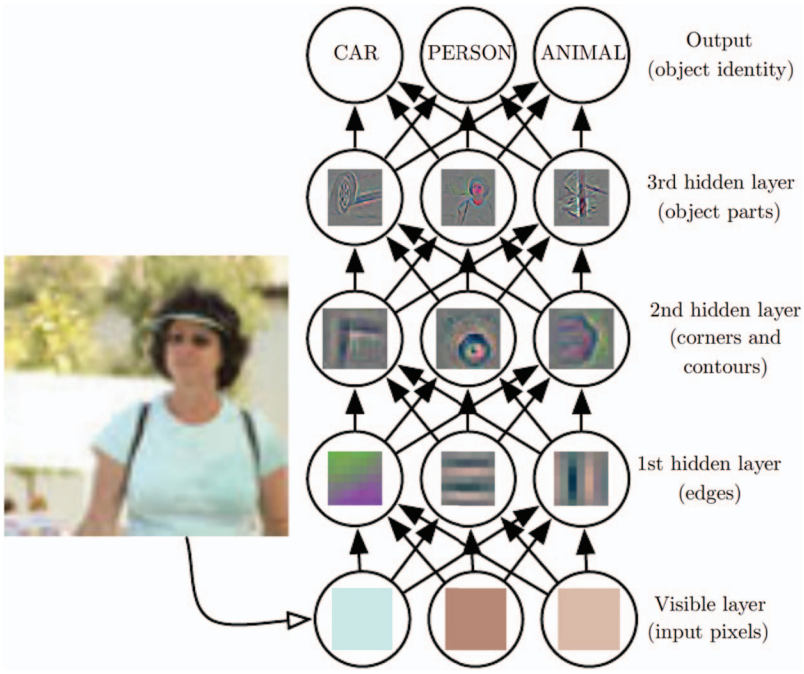
Prediction-focused studies like those just described can help inform intervention, prevention, and treatment endeavors in real-life scenarios. Focusing on prediction can also inform psychological theory by identifying important variables and relationships that can subsequently be investigated as potential causal factors underlying the phenomena being studied. However, psychologists often need to carefully select meaningful variables to include in machine learning models to get accurate predictions; this procedure is called *feature engineering* in machine learning (Zheng & Casari, 2018). The need for feature engineering makes it difficult to build accurate predictive models without a priori expertise in the phenomena being studied, which is often unavailable in complicated data sets with many variables. Indeed, the difficulty of feature engineering may be a strong contributor to the fact that no single method obtained consistently high predictive accuracy across the studies described above.

In this primer, we describe a potential solution to the feature engineering problem. Specifically, we introduce 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 more abstract concepts are built out of less abstract concepts. We visualize this information extraction process in Figure 1 (Goodfellow et al., 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* extract successively more and more abstract concepts, like edges or parts of objects. Finally, the model predicts the identity of the object in the image. For those familiar with structural equation modeling (SEM), hidden layers in deep learning models actually extract latent variables. However, unlike in SEM, these extracted latent variables may not be directly interpretable, may be related to each other and to the observed variables via nonlinear functional forms that are not known a priori, and are not subject to many of the assumptions and constraints required for most SEM estimators. Additionally, unlike traditional SEM and many classical statistical methods, deep learning algorithms *automatically* extract their own hidden layer/latent variable representations of the data to use for making predictions, thereby potentially avoiding the need for extensive feature engineering.

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 et al., 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 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., Karlsson et al., 2014; Sezer et al., 2020). Computer scientists have even employed ANNs

Figure 1

Illustration of a Deep Learning Model for Classifying Objects in Images



Note. 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. See the online article for the color version of this figure.

to accurately predict psychological constructs at unprecedented rates. For example, Suhara et al. (2017), Mikelsons et al. (2017), and Taylor et al. (2017) applied ANNs to forecast individuals' future moods using daily diary, wearable sensor, smartphone log, and weather data. Huang et al. (2018) combined two kinds of ANNs to predict bipolar individuals' mood disturbances using keystroke and circadian rhythm data. Aghaei et al. (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 et al., 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: (a) A lack of clarity about common terms like *deep learning* and fundamental concepts such as *artificial neural networks* (i.e., a *knowledge* barrier); (b) the complicated nature of building deep learning models in practice (i.e., a *complexity* barrier); and (c) a lack of standard, easy-to-use deep learning software (i.e., an *implementation* barrier). Because of space concerns, we primarily address the knowledge barrier in this primer and intend to more fully address the complexity and implementation barriers in future work.

The objective of this primer is to familiarize psychologists with deep learning concepts so that they can be better prepared to learn emerging 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 accomplish this goal via a two-pronged approach. First, we introduce psychologists to some of the benefits of using deep learning for predictive modeling. Second, we de-mystify deep learning by explaining common terms using both equations and words. 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 sequential data (e.g., daily diary data) and with image data (e.g., fMRI data) are explained. Specifically, we describe feedforward neural network (FNN), recurrent neural network (RNN), and convolutional neural network (CNN) models. Detailed toy examples and R code are presented for each model.

Benefits of Deep Learning for Predictive Modeling

Benefits Shared With Other Machine Learning Algorithms

Deep learning shares a number of benefits with machine learning algorithms in general. First, much like machine learning algorithms such as *random forests* (RFs; Breiman, 2001) and *support vector machines* (SVMs; Boser et al., 1992), deep learning algorithms implicitly model interactions and other nonlinearities that need to be explicitly specified in classical statistical methods such as linear regression (see Appendix A for a review of linear regression). This enables deep learning algorithms to obtain high predictive accuracy when the true causal relationships underlying the data are nonlinear (e.g., Yarkoni & Westfall, 2017). Second, deep learning effectively models multicollinear and high-dimensional data. Many classical statistical methods such as linear regression are unstable when the independent variables are multicollinear, often leading to inaccurate model predictions. Deep learning algorithms, on the other hand, may perform well with multicollinear and high-dimensional data sets by constructing highly predictive latent variable representations of the independent variables (De Veaux and Ungar, 1994). In this sense, deep learning algorithms are related to algorithms such as *principal components regression* (Hotelling, 1957; Kendall, 1957), *partial least squares* (Wold, 1966), and *penalized partial least squares* (Krämer et al., 2008), all of which aim to extract the most important information from the independent variables to make predictions.

A third benefit of the deep learning approach is that many strategies can be employed to prevent models from overfitting. A core objective of machine learning is to build predictive models that perform well on new, previously unseen data sets (e.g., Goodfellow et al., 2016; Hastie et al., 2009). Deep learning benefits from a number of techniques specifically developed to prevent deep learning models from overfitting (see Goodfellow et al., 2016, for a thorough, albeit technical, review). In Appendix B, we provide an overview of fundamental, general-purpose machine learning strategies for preventing overfitting. In particular, we explain the important concepts of changing a model's representational capacity, regularizing a model, and tuning a model's hyperparameters via validation set and *k*-fold cross-validation approaches. Familiarity with these fundamental concepts is essential for fitting deep learning models that perform well using previously unseen data sets. However, even if researchers never choose to fit a deep learning model, understanding and applying machine learning techniques such as regularization and cross-validation can increase the efficiency and reproducibility of many analysis pipelines without necessarily altering the end results (Yarkoni & Westfall, 2017).

Unique Benefits of Deep Learning Algorithms

Deep learning has some unique benefits in addition to those shared with general machine learning techniques. First, deep learning is capable of leveraging big data sets to obtain highly accurate predictions. Although all machine learning algorithms tend to benefit from increasing sample sizes, deep learning algorithms have been empirically found to leverage big data sets to increase their predictive accuracy beyond the accuracy attainable with other machine learning algorithms (e.g., RFs and SVMs; Goodfellow et al., 2016; LeCun et al., 2015). We anticipate 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. Deep learning algorithms may be able to capitalize on huge data sets such as observations from Internet sources (e.g., websites, blogs, and social media; Landers et al., 2016), cell phone behaviors (e.g., Hamaker & Wichers, 2017), and

genomic data sets (e.g., Plomin & Davis, 2009) to predict psychological outcomes with unprecedented accuracy.

We note, however, that "big data" are not *required* for deep learning models to obtain high predictive accuracy. Many tools exist for fitting accurate deep learning models using small data sets. For example, *transfer learning* (Pan & Yang, 2010; Torrey & Shavlik, 2009) is a machine learning technique wherein a model fitted to perform one task is repurposed to perform another task, typically to predict a different outcome on a new data set. Transfer learning often produces models with higher predictive accuracy than models fitted using only a single data set (e.g., Tan et al., 2018). Transfer learning may be useful for psychologists with small data sets, who could fit a deep learning model using a large, publicly available data set (e.g., a large-scale mental health survey data set) that is related to the small data set of interest (e.g., a small clinical sample), then "fine-tune" the model using the small data set. Recent work suggests that even without specialized approaches like transfer learning, deep learning models fitted using small data sets are capable of obtaining high predictive accuracy on new data (Olson et al., 2018).

Second, deep learning leverages small correlations for accurate predictions. Deep learning algorithms excel at discovering intricate relationships between large numbers of variables (LeCun et al., 2015). 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). Deep learning models, which are less readily interpretable than linear regression models, may perform well in such cases (although see Montavon et al. (2017) for a tutorial overview of methods for interpreting deep learning models).

Third, deep learning reduces the need for feature engineering. Psychologists usually need to think carefully about including meaningful, theoretically relevant variables in models to obtain accurate, generalizable predictions—that is, predictive modeling in psychology often requires careful feature engineering. In some application domains, deep learning algorithms can reduce the need for feature engineering by extracting their own (though usually not directly interpretable) representations of the independent variables to make predictions (LeCun et al., 2015). For example, the psychological analysis of text data (Iliev et al., 2015), image data, and video data (Barto et al., 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 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 personality characteristics (Mehta et al., 2020).

In a similar vein, deep learning may circumvent the need for measurement models. *Measurement models* refer to models that relate latent psychological constructs to *measures* or *indicators* of those constructs. Building measurement models is an essential but challenging part of conducting psychological research that typically requires careful consideration of many factors including exact specification of the relationships between constructs and indicators (e.g., Bollen, 2001). Oftentimes, psychologists construct measurement models to *explain* variability in some outcome variable of interest using latent predictor variables. In the case of item response theory (IRT), psychologists often build measurement models to obtain a score for an individual which can subsequently be used to make inferences pertaining to that individual (e.g., diagnostic status, performance ability). If, however, psychologists wish to accurately *predict* some outcome of interest rather than to make substantive inferences, deep learning may circumvent the need for measurement models by automatically extracting the latent variable representation of the input measures that is most predictive of the outcome variable.[1]

Fourth, deep learning algorithms often obtain higher predictive accuracy than other machine learning algorithms when the observations are sequences or images. We note that this does not imply that deep learning algorithms are not useful for research questions based in cross-sectional data (e.g., web-based behavioral measurements, cross-sectional survey data). Rather, deep learning algorithms outperform simpler machine learning algorithms more consistently and by a wider margin in sequence or image data sets than in cross-sectional data sets (e.g., Arik & Pfister, 2019; Goodfellow et al., 2016). Specifically, models called *recurrent neural networks* (RNNs) led to early advances in predictive modeling of text and speech data, whereas models called *convolutional neural networks* (CNNs) led to breakthroughs in processing image, video, and speech data (LeCun et al., 2015). Modern ANNs based on *residual neural networks* (ResNets; He et al., 2015), *temporal convolutional networks* (TCNs; Bai et al., 2018), and *transformers* (Vaswani et al., 2017) now achieve state-of-the-art performance on a variety of sequence and image modeling problems.

Fifth, deep learning can account for between- and within-group effects. Hierarchical (or nested) data structures in which multiple observations are taken from each of many clusters are common in psychology (e.g., students within schools, patients within clinics). Traditionally, hierarchical data have been analyzed using multilevel modeling (Raudenbush & Bryk, 2002). In deep learning, a similar approach called *multitask learning* fits models with parameters that are shared across groups as well as group-specific parameters (Caruana, 1997). Similar to how multilevel modeling improves on nonmultilevel statistical methods by permitting unbiased estimation with hierarchical data, multitask deep learning models may improve on nonmultitask machine learning and deep learning approaches by predicting psychological outcomes more accurately with hierarchical data (Zhang & Yang, 2017). For example, Taylor et al. (2017) found that a multitask deep learning approach outperformed both multitask SVMs and a hierarchical logistic regression approach at predicting mood, stress, and health status using self-reported behaviors as well as wearable sensor,

[1] Those familiar with principal components regression and its variant partial least squares may note that these methods provide a similar approach to deep learning by using causal indicators where the latent variables (or components) are predicted by the indicators. This contrasts with typical measurement models in SEM and IRT where reflective indicators are used—that is, where the indicators are predicted by the latent variables. Although deep learning does closely correspond to traditional methods based on causal indicators (for example, deep learning can be used to perform a nonlinear version of principal component analysis; Gorban et al., 2008), deep learning diverges from these methods by applying multiple nonlinear transformations to the extracted latent variables.

smartphone log, weather, and GPS data. See Bakker and Heskes (2004) for a discussion of the similarities between multitask learning and multilevel modeling.

Deep Learning Models, Terminology, and Notational Conventions

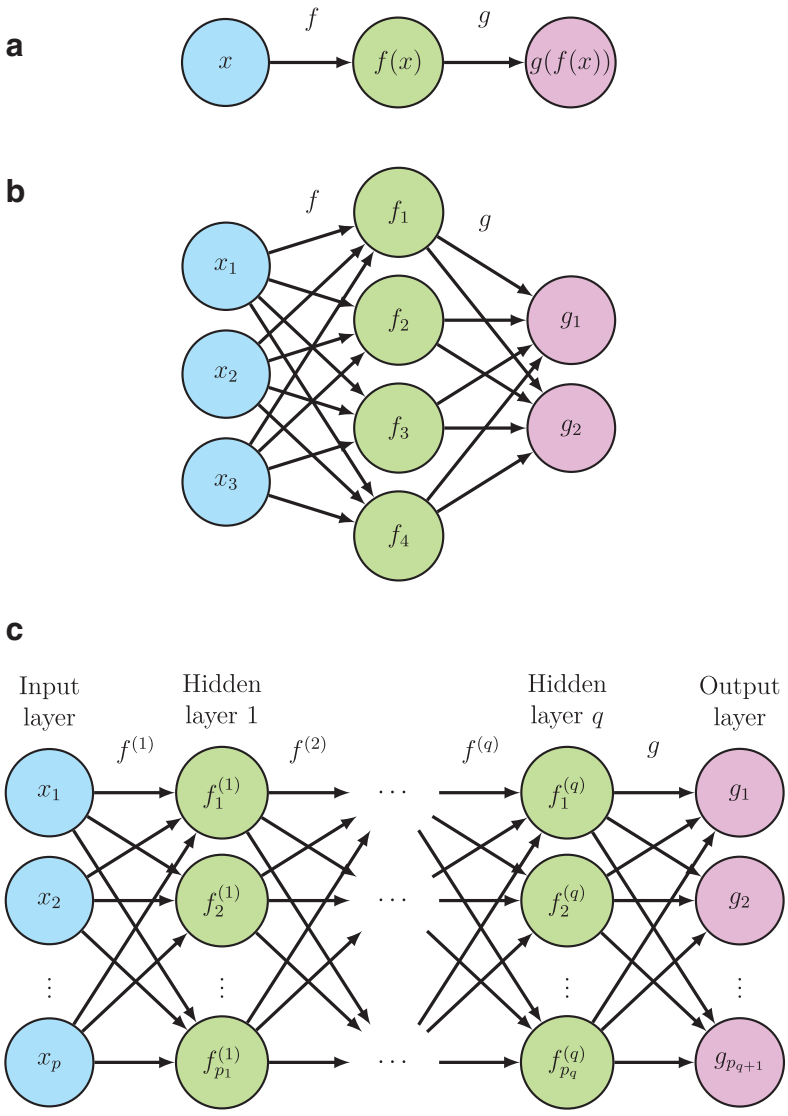
What exactly makes a machine learning model a *deep learning model*? A defining feature of most deep learning models is that they map the input observations through a sequence of functions where each function in the sequence is called a *layer*. In more technical terms, nearly all deep learning models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

$$g \circ f(x) = g(f(x)), \tag{1}$$

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 2a, we visualize this process with a schematic in which circles represent values and arrows represent functions.

The functions g and f in Equation 1 are *univariate* and *scalar-valued*, which means they take a single number as input and

Figure 2
Schematic Representations of Compositions of Functions



Note. (a) Composition of two scalar-valued functions. (b) Composition of two multivariate vector-valued functions. (c) Composition of $q + 1$ multivariate vector-valued functions. See the online article for the color version of this figure.

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

$$g \circ f(\mathbf{x}) = g(f(\mathbf{x})) \tag{2}$$

$$= [g_1([f_1(\mathbf{x}), \ \ldots, f_{p_1}(\mathbf{x})]^\top), \ \ldots, g_{p_2}([f_1(\mathbf{x}), \ \ldots, f_{p_1}(\mathbf{x})]^\top)]^\top, \tag{3}$$

where $\mathbf{x}$ is a $p \times 1$ vector, $f(\mathbf{x})$ is a $p_1 \times 1$ vector, and $g(f(\mathbf{x}))$ is a $p_2 \times 1$ vector. Equation 3 demonstrates that any vector-valued function actually consists of many scalar-valued functions. For example, the vector-valued function f actually consists of p_1 different scalar-valued functions $f_1, \ldots, f_{p_1}$ that each take in the input vector $\mathbf{x}$ and spit out the single numbers $f_1(\mathbf{x}), \ \ldots, \ f_{p_1}(\mathbf{x})$. We visualize a composition of two multivariate vector-valued functions in Figure 2b. Notice that p, p_1, and p_2 (i.e., the number of elements in $\mathbf{x}$, $f(\mathbf{x})$, and $g(f(\mathbf{x}))$, respectively) are not necessarily equal: In Figure 2b, $\mathbf{x}$ is three-dimensional (i.e., $p = 3$), $f(\mathbf{x})$ is four-dimensional (i.e., $p_1 = 4$), and $g(f(\mathbf{x}))$ is two-dimensional (i.e., $p_2 = 2$). 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

$$g \circ f^{(2)} \circ f^{(1)}(\mathbf{x}) = g(f^{(2)}(f^{(1)}(\mathbf{x}))), \tag{4}$$

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

$$g \circ f^{(q)} \circ \ \ldots \ \circ f^{(2)} \circ f^{(1)}(\mathbf{x}) = g(f^{(q)}(\ \ldots \ f^{(2)}(f^{(1)}(\mathbf{x})))), \tag{5}$$

where $\mathbf{x}$ is a $p \times 1$ vector, $f^{(i)}(\ \ldots f^{(1)}(\mathbf{x}))$ is a $p_i \times 1$ vector for $i = 1, \ \ldots, \ q$, and $g(\ \ldots f^{(1)}(\mathbf{x}))$ is a $p_{q+1} \times 1$ vector. Nearly all deep learning models are compositions of many multivariate vector-valued functions as in Equation 5, where each function has parameters we can optimize to help the model accurately predict the outcome variable.

Equation 5 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 ith 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 ith layer, denoted p_i, is called the *width* of layer i. The process in Equation 5 is visualized with annotated layers in Figure 2c. 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).

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 fit or models that do not perform well. *Artificial neural networks* (ANNs) are one popular solution to this problem. ANNs are a broad class of nonlinear statistical models that can be composed together in many layers. They were initially inspired by biological neural mechanisms (McCulloch & Pitts, 1943; McClelland et al., 1986; 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. The earliest ANNs only had one hidden layer, but deep learning implementations of ANNs have multiple hidden layers. ANNs have been extremely successful under the deep learning paradigm because they can be efficiently fitted (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). Like linear regression, FNNs are useful for predicting outcomes of interest using tabular data sets (i.e., data sets which can be formatted as the standard $N \times p$ design matrix $\mathbf{X}$ used in linear regression; see Appendix A). Next, we discuss recurrent neural networks (RNNs), which were developed to predict outcomes of interest using sequential data (e.g., daily diary data; physiological trace data). Although RNNs are often outperformed in practice by convolutional neural networks and other modern ANNs (Bai et al., 2018; Vaswani et al., 2017), familiarity with RNN basics is an excellent starting point for exploring more recent deep learning approaches to sequence modeling. Finally, we discuss convolutional neural networks (CNNs), which are powerful tools for predicting outcomes with both image data (e.g., fMRI data) and with sequential data. In this primer, we focus on applications of CNNs to image data because of space concerns, but we note that applications of CNNs to sequential data are conceptually similar. We compare linear regression, FNNs, RNNs, and CNNs in Table 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 $g(f(\mathbf{x}))$ as in Equation 2. It can be used either for *regression* (i.e., predicting continuous outcomes) or for *classification* (i.e., predicting categorical outcomes).

Just like in linear regression, the single-layer FNN starts with N observations of p predictor variables collected in an $N \times p$ design matrix $\mathbf{X}$ as well as N observations of one outcome variable collected in an $N \times 1$ vector $\mathbf{y}$. In linear regression, we typically denote the ith observed vector of predictor variables (i.e., a single

Table 1
Comparison of Typical Use Cases for the Linear Regression Model and Three Basic Artificial Neural Network Models

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

[a] High-dimensional (e.g., for tabular data, the number of variables is larger than the sample size).

row of $\mathbf{X}$) as $\mathbf{x}_i$ and the ith observed value of the outcome variable as y_i for $i = 1, \ldots, N$. In this primer, we simplify notation by dropping all i subscripts so that $\mathbf{x}_i$ is written as $\mathbf{x}$ and y_i is written as y. This simplification is justified because single-layer FNNs and other ANNs are usually fitted 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 produce a $p_1 \times 1$ hidden layer representation $\mathbf{h}$ of each $p \times 1$ observation $\mathbf{x}$ that can subsequently be used to predict the corresponding outcome y. To produce this hidden layer representation, the single-layer FNN first computes p_1 different weighted sums of the observed predictor values x_j plus an intercept:

$$s_k = b_{k,0}^{\text{hx}} + \sum_{j=1}^{p} b_{k,j}^{\text{hx}} x_j, \quad k = 1, \ldots, p_1, \tag{6}$$

where s_k are weighted sums called *activations*, $b_{k,j}^{\text{hx}}$ are weight parameters, $b_{k,0}^{\text{hx}}$ are intercept parameters, and p_1 is the number of hidden layer nodes. Equation 6 is simply a linear regression model (see Equation A.1) 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 "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 et al., 2015).

Next, the single-layer FNN applies a nonlinear function f to each activation s_k to compute the *hidden layer* or the *derived predictor* nodes:

$$h_k = f(s_k), \quad k = 1, \ldots, p_1. \tag{7}$$

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 $g(f(\mathbf{x}))$ model described in Equation 2. The nonlinear function f is called an *activation function* and enables the single-layer FNN to model outcomes that vary nonlinearly 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

$$f(z) = \frac{1}{1 + e^{-z}}, \tag{8}$$

although modern single-layer FNNs mostly use the *rectified linear unit* or *ReLU* (Glorot et al., 2011; Jarrett et al., 2009; Nair & Hinton, 2010)

$$f(z) = \max(0, z). \tag{9}$$

The sigmoid activation function is visualized in Figure 3a, and the ReLU activation function is visualized in Figure 3b. 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 outcome y. To do so, it computes a weighted sum of the hidden layer values plus an intercept term, then applies another nonlinear activation function g to this sum:

$$y = g\left(b_0^{\text{yh}} + \sum_{k=1}^{p_1} b_k^{\text{yh}} h_k\right) + \varepsilon, \tag{10}$$

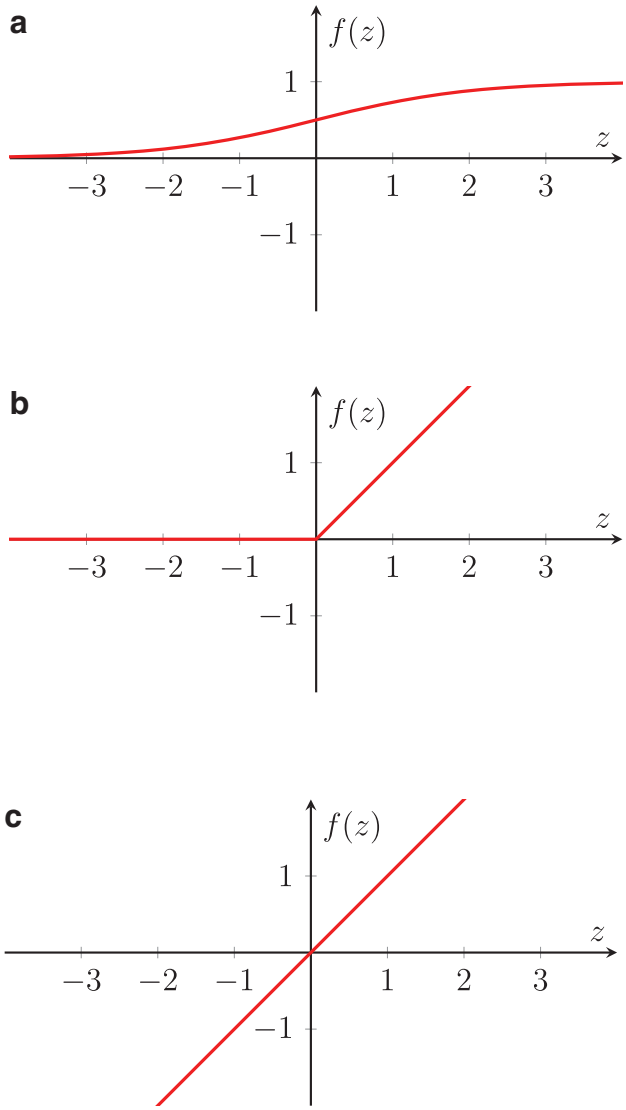
where "yh" superscripts indicate weight parameters to the output node from the hidden layer nodes and ε is a random error term. The single-layer FNN is visualized in Figure 4. $\hat{y}$, the predicted output value in the final node, is plugged in to an objective function (e.g., mean squared error) to 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 SEM, where it is used to include intercepts in path diagrams, often as triangles (Bollen, 1989).

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

$$g(z) = z. \tag{11}$$

Figure 3c 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 outcome y that may be either zero (corresponding to the first output category) or one (corresponding to the second output category). Here, we choose g to be the sigmoid function (Equation 8). Figure 3a 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 a $k \times 1$ outcome vector $\mathbf{y}$ that is dummy coded such that the ith dummy variable is equal to 1 and all other dummy variables are equal to 0 (i.e., $\mathbf{y} = [0, \ldots, 0, 1, 0, \ldots, 0]^\top$). In this case, we choose g to be the *softmax* function

Figure 3
Common Activation Functions Used in Artificial Neural Network Models



Note. (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. See the online article for the color version of this figure.

$$g(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}, i = 1, \ldots, k, \tag{12}$$

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 ith 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:

$$y = g\left((\mathbf{b}^{\text{yh}})^\top \mathbf{h} + b_0^{\text{yh}}\right) + \varepsilon, \tag{13}$$

$$\mathbf{h} = f\left(\mathbf{B}^{\text{hx}}\mathbf{x} + \mathbf{b}_{\mathbf{0}}^{\text{hx}}\right), \tag{13a}$$

where

$$\underset{p_1 \times 1}{\mathbf{b}^{\text{yh}}} = \begin{bmatrix} b_1^{\text{yh}} \\ b_2^{\text{yh}} \\ \vdots \\ b_{p_1}^{\text{yh}} \end{bmatrix}, \underset{p_1 \times 1}{\mathbf{h}} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_{p_1} \end{bmatrix}, \tag{13b-c}$$

$$\underset{p_1 \times p}{\mathbf{B}^{\text{hx}}} = \begin{bmatrix} b_{1,1}^{\text{hx}} & b_{1,2}^{\text{hx}} & \cdots & b_{1,p}^{\text{hx}} \\ b_{2,1}^{\text{hx}} & b_{2,2}^{\text{hx}} & \cdots & b_{2,p}^{\text{hx}} \\ \vdots & \vdots & \ddots & \vdots \\ b_{p_1,1}^{\text{hx}} & b_{p_1,2}^{\text{hx}} & \cdots & b_{p_1,p}^{\text{hx}} \end{bmatrix}, \underset{p \times 1}{\mathbf{x}} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \end{bmatrix}, \underset{p_1 \times 1}{\mathbf{b}_{\mathbf{0}}^{\text{hx}}} = \begin{bmatrix} b_{1,0}^{\text{hx}} \\ b_{2,0}^{\text{hx}} \\ \vdots \\ b_{p_1,0}^{\text{hx}} \end{bmatrix}, \tag{13d-f}$$

and the activation functions f and g are applied to vectors element-wise. 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. As noted in the introduction, $\mathbf{h}$ is in fact a vector of p_1 *latent variables*, or unobserved variables that summarize the information contained in the original observation. Latent variables are likely familiar to psychologists from SEM, which aims to explain the relationships between a set of observed variables in terms of a number of unobserved latent constructs. In single-layer FNNs, however, each latent variable in $\mathbf{h}$ depends on every observed variable in $\mathbf{x}$. This may make the latent variables in single-layer FNNs more difficult to interpret than those in SEM, where each latent variable is interpretable in part because it is only related to a subgroup of conceptually similar observed variables.

Single-Layer Feedforward Neural Networks Generalize Linear Regression

Previously, we mentioned that ANNs can be thought of as generalizations of linear regression. We now use basic algebra to show that the single-layer FNN can be reduced to the linear regression model. We first choose the single-layer FNN's activation functions g and f to be identity functions:

$$g(z) = z, \; f(z) = z. \tag{14}$$

Our single-layer FNN can then be written as

$$y = b_0^{\text{yh}} + \sum_{k=1}^{p_1} b_k^{\text{yh}} h_k + \varepsilon, \tag{15}$$

$$h_k = b_{k,0}^{\text{hx}} + \sum_{j=1}^{p} b_{k,j}^{\text{hx}} x_j, \; k = 1, \ldots, p_1. \tag{15a}$$

We can substitute Equation 15a into Equation 15 and rearrange to obtain

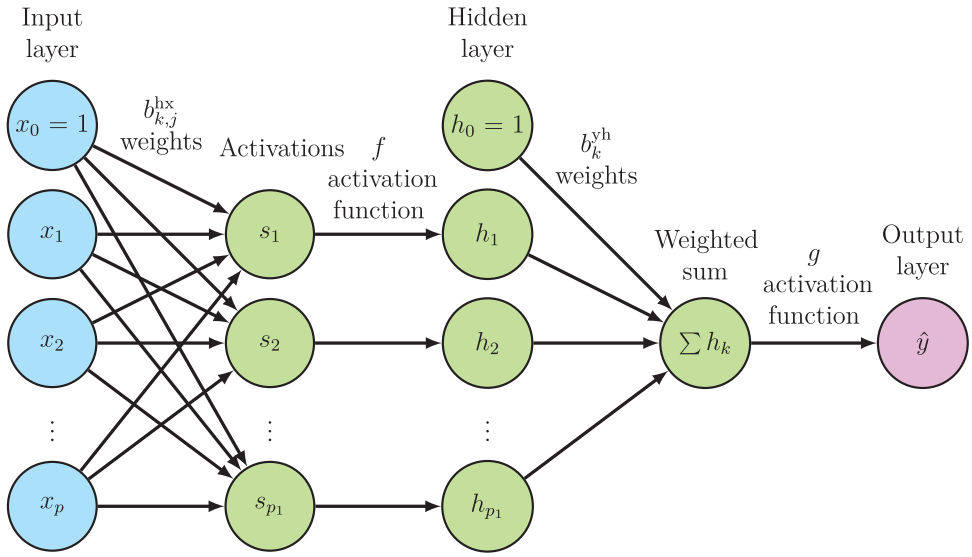
$$b_0^{\text{yh}} + \sum_{k=1}^{p_1} b_k^{\text{yh}} \left(b_{k,0}^{\text{hx}} + \sum_{j=1}^{p} b_{k,j}^{\text{hx}} x_j \right) + \varepsilon \tag{16}$$

$$= b'_0 + \sum_{j=1}^{p} b'_j x_j + \varepsilon, \tag{17}$$

where

Figure 4

Schematic Representation of the Single-Layer Feedforward Neural Network



Note. See the online article for the color version of this figure.

$$b'_0 = b^{yh}_0 + \sum_{k=1}^{p_1} b^{yh}_k b^{hx}_{k,0},\ b'_j = \sum_{k=1}^{p_1} b^{yh}_k b^{hx}_{k,j},\ j = 1, \ldots, p. \tag{17a}$$

Equation 17 clearly has the same form as the linear regression model in Equation A.1.[2] 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 "yh" and "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 13b, 13d, and 13f to count parameters and Equations 17a to count equations). Infinitely many sets of single-layer FNN parameters will satisfy Equations 17a 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 (e.g., Montavon et al., 2017). These methods aim to interpret the concepts learned by the ANN's hidden layers and to identify the most important predictor variables. We now turn to a detailed toy example to illustrate how single-layer FNNs may be useful for psychological research despite being challenging to interpret.

Single-Layer FNN Toy Example: Predicting Alcohol Use Disorder Using Cross-Sectional Survey Data

Imagine that we have access to data from a large, nationally representative, cross-sectional survey that was designed to measure mental health and substance use. Such data are often publicly available (e.g., the National Survey on Drug Use and Health; https://nsduhweb.rti.org/respweb/homepage.cfm) and may contain a wealth of information about various psychological phenomena in the population at large. Our survey includes the demographic variables gender and age, diagnostic criteria for major depressive disorder (MDD), and diagnostic criteria for nicotine, marijuana, and alcohol use disorders (NUDs, MUDs, and AUDs, respectively). We are interested in finding out whether meeting diagnostic criteria for AUD can be predicted using the other demographic, mental health, and substance use variables available in our data set. This question may be interesting in clinical psychology, for example, where researchers and clinicians may wish to provide targeted interventions for individuals who are identified as being at risk for problematic outcomes such as AUD.

Assume that our data set includes $N = 5{,}000$ individuals. Our AUD outcome is a binary variable that is equal to one when an individual meets the relevant diagnostic criteria and is equal to

[2] When we use a suitable optimization procedure, our estimates for the $b'_0, b'_1, \ldots, b'_p$ parameters will converge to the $b_0, b_1, \ldots, b_p$ parameter estimates produced by the usual linear regression algorithm (see Equation A.6; e.g., Baldi & Hornik, 1995).

zero otherwise. Our gender, MDD, NUD, and MUD predictor variables are also binary. Our initial age predictor is a number greater than or equal to 18 (i.e., all individuals are adults); we normalize age by subtracting this predictor's minimum value and dividing by its range before model fitting. Our goal is to use each individual's predictors $\mathbf{x}$ to predict whether they meet diagnostic criteria for AUD (i.e., their outcome value is one) or not (i.e., their outcome value is zero). Prediction problems like this are called classification problems in machine learning because the goal is to assign each individual to the correct class (i.e., meeting diagnostic criteria for AUD or not). We will develop a predictive model using a training-test split approach (see Appendix B) wherein models are fitted using a training set of 4,500 observations and model predictive accuracy is evaluated using a test set consisting of the remaining 500 observations.

We used R (R Core Team, 2020) to simulate predictors based on the description in the previous paragraph. Specifically, predictors were sampled for each individual as follows:

$$\text{gender} \sim \text{Bernoulli}(0.5), \tag{18}$$

$$\text{age} \sim \mathcal{U}(18, 85), \tag{19}$$

$$\text{MUD} \sim \text{Bernoulli}(f(-1 - \text{age} + 2 \cdot \text{gender})), \tag{20}$$

$$\text{NUD} \sim \text{Bernoulli}(f(-1 - \text{age} + 2 \cdot \text{gender} + 0.5 \cdot \text{MUD})), \tag{21}$$

$$\text{MDD} \sim \text{Bernoulli}(f(-1 - \text{age} + 2 \cdot \text{gender} + 0.5 \cdot \text{MUD} + 0.5 \cdot \text{NUD})), \tag{22}$$

where $\mathcal{U}(a, b)$ denotes a uniform distribution with lower bound a and upper bound b and f denotes the sigmoid function (Equation 8). This sampling scheme was motivated as follows: (a) Approximately half of the sampled individuals should be male (gender = 1; Equation 18); (b) individuals should all be adults (i.e., over age 18) and all age groups should be evenly represented in the sample (Equation 19); (c) holding other variables constant, individuals should be at relatively low risk of MDD and substance use disorders (SUDs; intercept of −1 in Equations 20–22); (d) increasing age should protect against risk of MDD and SUDs (negative coefficient on age in Equations 20–22); (e) being male should increase risk of MDD and SUDs (positive coefficient on gender in Equations 20–22); and (f) having any of MDD, MUD, or NUD should increase risk for the other variables (positive coefficients on MDD, MUD, and NUD in Equations 20–22). AUD outcome values were then generated nonlinearly from the predictor values where all predictors related to the outcome to varying degrees:

$$\begin{aligned}\text{AUD} \sim \text{Bernoulli}(f(&-1 - (\text{age} + \text{age}^2 + \text{age}^3 + \text{age}^4 + \text{age}^5) \\ &+ 2 \cdot \text{gender} + 0.5 \cdot \text{MDD} + 0.5 \cdot \text{NUD} + 0.5 \cdot \text{MUD} \\ &- 2 \cdot (\text{age} + \text{age}^2 + \text{age}^3 + \text{age}^4 + \text{age}^5) \cdot \text{gender})),\end{aligned} \tag{23}$$

where the age predictor was normalized prior to generation. Equation 23 is the same as Equations 20–22 except age now enters the model nonlinearly and interacts with gender such that older males have decreased risk of AUD. Approximately 26% of simulated individuals met AUD criteria (i.e., 26% of the generated AUD outcome values were equal to one).

Once the data were simulated, we used the R interface to Keras (a Python package for fitting ANNs; Falbel et al., 2019) to classify individuals using the single-layer FNN given in Equations 13 and 13a. We first constructed the model as follows:

```
fnn = keras_model_sequential()
fnn %>%
  layer_dense(units = 5,
              activation = 'relu',
              input_shape = c(5)) %>%
  layer_dense(units = 1,
              activation = 'sigmoid')
```

In Keras, ANNs are constructed one layer at a time. The above code first creates a single hidden layer (i.e., a `layer_dense()`) with five nodes and a ReLU activation function then connects these hidden nodes to the outcome node and applies a sigmoid activation function. Next, we specified some parameters for model fitting:

```
fnn %>% compile(
  loss = 'binary_crossentropy',
  optimizer = optimizer_adam(),
  metrics = c('accuracy')
)
```

Here, we specify the objective function or *loss function* to be the log-likelihood of the Bernoulli-distributed AUD outcome summed over all individuals. This log-likelihood is called the *binary cross-entropy* in machine learning and is the same objective function used in logistic regression (Kutner et al., 2004). By setting `optimizer = optimizer_adam()`, we fit our model using an optimization procedure called Adam (Kingma & Ba, 2015). Adam demonstrates good performance in practice with little to no hyperparameter tuning (Goodfellow et al., 2016). We also choose to assess model performance using predictive accuracy (i.e., the proportion of outcome values correctly predicted by the model).

Finally, we fitted the model as follows:

```
fnn %>% fit(
  as.matrix(train_data$X),
  as.matrix(train_data$y),
  epochs = 70,
  batch_size = 128
)
fnn %>% evaluate(as.matrix(test_data$X),
                 as.matrix(test_data$y))
```

The first two inputs to the `fit()` function are a test set matrix of predictors and the associated vector of outcomes, respectively. ANNs are fitted iteratively using small random samples of observations at each iteration. We randomly sample 128 observations at each fitting iteration using `batch_size = 128`. If we sample observations without replacement, we will eventually have sampled all the observations in our data set and must replace all the observations before fitting can proceed. A single full pass through every observation in the data set is called an *epoch* in deep learning. Our code continues fitting until we have sampled all the observations in the data set 70 times by setting `epochs = 70`. We compute the fitted model's predictive accuracy using the `evaluate()` function; our fitted model obtained a predictive accuracy of 0.86 the test set. An important note regarding these analyses is that we did not tune our model hyperparameters. A complete R script for our sim-

ulation and analysis is available as online supplemental material. Our model is visualized in Figure 5.

We now step through how our model computes a predicted value for a single observation from our simulated data set using our fitted model parameters. This observation's normalized age was 0.22, and all of the other (binary) predictors were 1. We first compute the vector of activations $\mathbf{s}$ by multiplying our vector of predictors by a weight matrix $\mathbf{B}^{\text{hx}}$ and adding an intercept vector $\mathbf{b}_{\mathbf{0}}^{\text{hx}}$ just like in linear regression:

$$\mathbf{s} = \mathbf{B}^{\text{hx}}\mathbf{x} + \mathbf{b}_{\mathbf{0}}^{\text{hx}} \tag{24}$$

$$= \begin{bmatrix} -0.03 & 0.93 & 0.47 & 0.07 & 0.50 \\ -0.29 & -2.00 & 1.59 & 0.16 & -1.79 \\ -0.01 & 0.26 & 0.81 & -0.13 & 0.57 \\ 0.48 & 0.31 & 0.57 & -0.52 & 0.44 \\ -0.03 & 0.02 & -0.07 & 0.05 & 0.42 \end{bmatrix} \begin{bmatrix} 1 \\ 0.22 \\ 1 \\ 1 \\ 1 \end{bmatrix} + \begin{bmatrix} -0.17 \\ 0.13 \\ 0.52 \\ -0.28 \\ 0.43 \end{bmatrix} \tag{25}$$

$$= [1.05 \quad -0.65 \quad 1.82 \quad 0.76 \quad 0.80]^{\top}. \tag{26}$$

We then apply the ReLU activation function f (Equation 9) to our vector of activations $\mathbf{s}$ to compute our hidden layer vector of latent variables:

$$\mathbf{h} = f(\mathbf{s}) = [1.05 \quad 0.00 \quad 1.82 \quad 0.76 \quad 0.80]^{\top}, \tag{27}$$

which simply sets any negative values in $\mathbf{s}$ to zero. We then repeat the process with $\mathbf{h}$: We multiply it by a weight vector $(\mathbf{b}^{\text{yh}})^{\top}$, add an intercept b_0^{yh}, then apply the sigmoid activation function g (Equation 8):

$$y = g\big((\mathbf{b}^{\text{yh}})^{\top}\mathbf{h} + b_0^{\text{yh}}\big) \tag{28}$$

$$= g\left(\begin{bmatrix} 0.48 & 1.31 & -0.84 & 0.62 & 1.57 \end{bmatrix} \begin{bmatrix} 1.05 \\ 0.00 \\ 1.82 \\ 0.76 \\ 0.80 \end{bmatrix} - 1 \right)$$

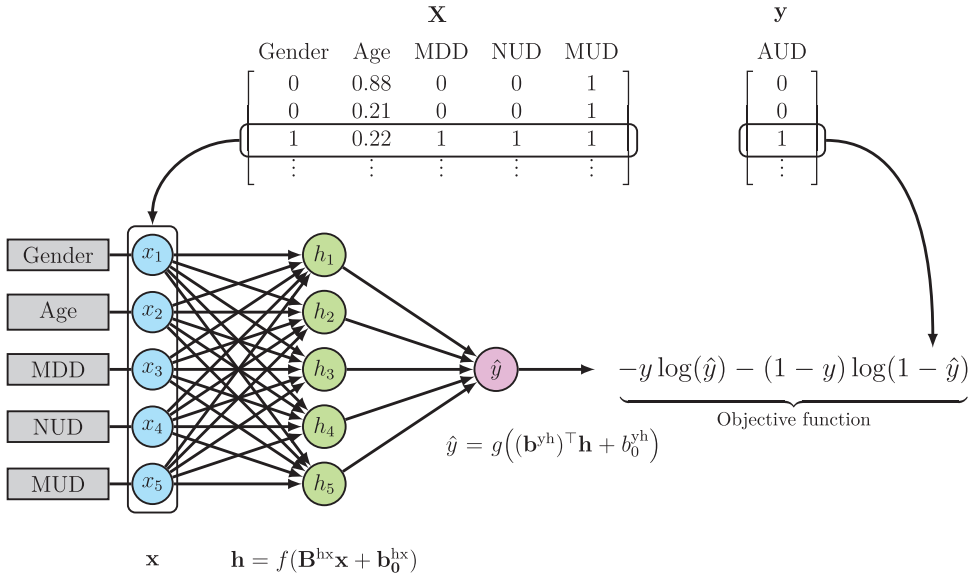
$$= g(-0.29) = 0.43. \tag{29}$$

The outputted value of 0.43 corresponds to the predicted probability that the simulated individual meets AUD criteria.

Recall that our single-layer FNN's objective function (see Figure 5) is the same objective function used in logistic regression. To illustrate the difference between these methods, we compared our model's performance to that of logistic regression with a lasso penalty fitted using the R package glmnet (Friedman et al., 2010; see McNeish, 2015, for an overview of lasso penalized regression). Predictive accuracy as well as sensitivity (i.e., the proportion of individuals meeting AUD criteria who were correctly predicted by the model) and specificity (i.e., the proportion of individuals not meeting AUD criteria who were correctly predicted by the model) were computed on the test set for both models. Our single-layer FNN (accuracy = 0.86, sensitivity = 0.62, specificity = 0.94) mostly outperformed penalized logistic regression (accuracy = 0.80, sensitivity = 0.30, specificity = 0.97), suggesting that the FNN better cap-

Figure 5

Schematic Visualizing How Single-Layer Feedforward Neural Networks May Be Applied to Cross-Sectional Survey Data



Note. Age is normalized such that its values range from zero to one. AUD = alcohol use disorder; MDD = major depressive disorder; MUD = marijuana use disorder; NUD = nicotine use disorder. See the online article for the color version of this figure.

tured the underlying nonlinear relationship between the predictors and the outcomes. Additionally, the FNN obtained this superior performance with no hyperparameter tuning. Although ANNs often obtain reasonable performance with little to no hyperparameter tuning, conducting extensive hyperparameter tuning may produce astonishing performance gains. See Bengio (2012), Heaton (2008), or Smith (2018) for discussions of hyperparameter tuning for ANNs.

We note that complex survey designs typically include sampling weights to ensure that population subgroups are adequately represented in subsequent analyses (Lavalleé and Beaumont, 2015). We chose not to simulate sampling weights to facilitate our comparison of single-layer FNNs with lasso penalized logistic regression. However, if we had simulated sampling weights, they could have been included in our single-layer FNN objective function simply by multiplying each individual's objective function value by their sampling weight. This is easy to do using Keras—a vector of sampling weights can be included in `fit()` using the `sample_weight` argument.

Deep Feedforward Neural Networks

Single-layer FNNs are not deep learning models because they only have one hidden layer. *Deep feedforward neural networks*, also called *multilayer perceptrons*, extend single-layer FNNs by including more than one hidden layer. Using many hidden layers may allow deep FNNs to model complicated, nonlinear 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. To produce its first p_1 hidden layer nodes, the deep FNN computes p_1 weighted sums of the predictor values x_j plus an intercept, then applies an activation function $f^{(1)}$ to each sum:

$$h_k^{(1)} = f^{(1)}\left(b_{k,0}^{\mathrm{h_1x}} + \sum_{j=1}^{p} b_{k,j}^{\mathrm{h_1x}} x_j\right),\ k = 1,\ \ldots,p_1 \tag{30}$$

where "h_{1x}" superscripts indicate to weight parameters to the first hidden layer nodes from the predictor nodes. Equation 30 is clearly the same as Equation 7 substituted into Equation 6 – 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 then 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:

$$h_k^{(\ell)} = f^{(\ell)}\left(b_{k,0}^{\mathrm{h_\ell h_{\ell\text{-}1}}} + \sum_{j=1}^{p_{\ell-1}} b_{k,j}^{\mathrm{h_\ell h_{\ell\text{-}1}}} h_j^{(\ell-1)}\right),\ k = 1,\ \ldots,p_\ell,\quad \ell = 2,\ \ldots,q, \tag{31}$$

where q denotes the model depth, p_ℓ denotes the number of nodes at hidden layer ℓ, and "$\mathrm{h_\ell h_{\ell\ -\ 1}}$" superscripts indicate weight parameters to hidden layer ℓ nodes from hidden layer ℓ-1 nodes.

The outcome node is predicted by taking a weighted sum of the final hidden layer nodes plus an intercept term, then applying an activation function g to this sum:

$$y = g\left(b_0^{\mathrm{yh_q}} + \sum_{k=1}^{p_q} b_k^{\mathrm{yh_q}} h_k^{(q)}\right) + \varepsilon, \tag{32}$$

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

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

$$y = g\left((\mathbf{b}^{\mathrm{yh_q}})^\top \mathbf{h}^{(q)} + b_0^{\mathrm{yh_q}}\right) + \varepsilon, \tag{33}$$

$$\mathbf{h}^{(\ell)} = f^{(\ell)}\left(\mathbf{B}^{\mathrm{h_\ell h_{\ell-1}}} \mathbf{h}^{(\ell-1)} + \mathbf{b_0}^{\mathrm{h_\ell h_{\ell-1}}}\right),\ \ell = 2,\ \ldots,q, \tag{33a}$$

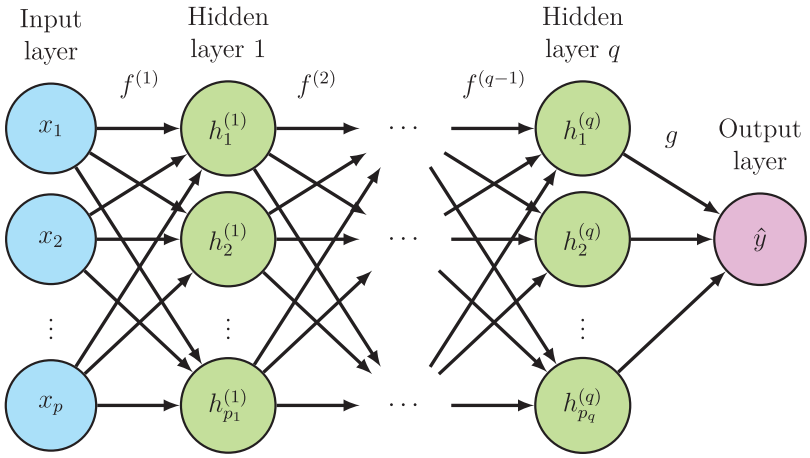
$$\mathbf{h}^{(1)} = f\left(\mathbf{B}^{\mathrm{h_1x}} \mathbf{x} + \mathbf{b_0}^{\mathrm{h_1x}}\right), \tag{33b}$$

where $\mathbf{x}$ is the $p \times 1$ vector of predictors, $\mathbf{B}^{\mathrm{h_1x}}$ is the $p_1 \times p$ weight matrix from the predictors to the first hidden layer, $\mathbf{b}_0^{\mathrm{h_1x}}$ is the $p_1 \times 1$ intercept vector from the predictors to the first hidden layer, $\mathbf{h}^{(\ell)}$ is the $p_\ell \times 1$ hidden layer representation at layer ℓ, $\mathrm{B}^{\mathrm{h_\ell h_{\ell-1}}}$ weight matrix from hidden layer $\ell - 1$ to hidden layer ℓ, $\mathbf{b_0}^{\mathrm{h_\ell h_{\ell-1}}}$ is the $p_\ell \times 1$ intercept vector from hidden layer $\ell - 1$ to hidden layer ℓ, $\mathbf{b}^{\mathrm{yh_q}}$ is the $p_q \times 1$ wt vector from the final hidden layer to the outcome, and $b_0^{\mathrm{yh_q}}$ is the intercept from the final hidden layer to the outcome. We visualize the deep FNN in Figure 6. 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 the *universal approximation theorem*, which holds that both single-layer and deep FNNs are capable of approximating a wide variety of 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 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. Deep FNNs with many hidden layers may need fewer nodes at each layer to learn the true f 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.

[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 et al., 2017; Sonoda and Murata, 2017) do not require the activation functions to be bounded.

Figure 6
Schematic Representation of the Deep Feedforward Neural Network



Note. See the online article for the color version of this figure.

Recurrent Neural Networks

Recurrent neural networks (RNNs) are specialized ANNs for processing sequential data where the order of the observations is meaningful. Like FNNs, RNNs can be used for regression or classification. Although RNNs are outperformed by CNNs and more modern ANNs in some practical settings, this section illustrates fundamental concepts that are applicable to a broad range of ANN approaches to modeling sequential data.

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, which we denote as $\boldsymbol{\mathcal{X}}$, typically contain many sequences. For example, daily diary data sets typically include N different individuals, each of whom is asked p questions each day for T_i days, $i = 1, \ldots, N$. 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. We write 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 predictor values observed for individual i at time point t. In practical applications of RNNs, it is common to first ensure that all sequences are the same length (e.g., by appending vectors of zeros to short sequences), say length T, then to collect each individual's sequences into a $T \times p$ matrix $\mathbf{X}_i$. Our data set may then be thought of as a collection of N $T \times p$ matrices or, equivalently, as an $N \times T \times p$ array called a *tensor*. Tensors, which are multidimensional arrays that generalize matrices to higher-dimensions, are an important concept because deep learning software typically requires data to be input in tensor form. Tensors are briefly discussed and visualized in the Convolutional Neural Networks section.

We can now formulate the simple RNN model. As before, we drop all i subscripts when specifying models. 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 produce 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 the previous time step when making predictions. Intuitively, the simple RNN is just a single-layer FNN with loops carrying information forward through time. The simple RNN models the current outcome node as a function of the current predictor nodes as well as the previous hidden layer nodes:

$$y^{(t)} = g\left(b_0^{\text{yh}} + \sum_{k=1}^{p_1} b_k^{\text{yh}} h_k^{(t)}\right) + \varepsilon^{(t)}, \; t = 1, \ldots, T, \tag{34}$$

$$h_k^{(t)} = f\left(b_{k,0}^{\text{hx}} + \sum_{j=1}^{p} b_{k,j}^{\text{hx}} x_j^{(t)} + \sum_{k=1}^{p_1} b_k^{\text{hh}} h_k^{(t-1)}\right), \; k = 1, \ldots, p_1, \quad t = 1, \ldots, T, \tag{34a}$$

where "hh" superscripts indicate *recurrent weight parameters* connecting hidden layer nodes at subsequent time steps, $\varepsilon^{(t)}$ is the error at time point t, and the initial hidden layer nodes $h_k^{(0)}$ are all defined to be 0. Equations 34 and 34a 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 reuses the same weight parameters at each time step. Reusing 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:

$$y^{(t)} = g\left((\mathbf{b}^{\text{yh}})^\top \mathbf{h}^{(t)} + b_0^{\text{yh}}\right) + \varepsilon^{(t)}, \; t = 1, \ldots, T, \tag{35}$$

$$\mathbf{h}^{(t)} = f\left(\mathbf{B}^{\text{hx}} \mathbf{x}^{(t)} + \mathbf{B}^{\text{hh}} \mathbf{h}^{(t-1)} + \mathbf{b}_{\mathbf{0}}^{\text{hx}}\right), \; t = 1, \ldots, T, \tag{35a}$$

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, $\mathbf{b}_0^{\text{hx}}$ 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 $\mathbf{h}^{(\mathbf{0})}$ are defined to be 0 (a $p_1 \times 1$ vector of zeros). Just like Equations 34 and 34a, Equations 35 and 35a 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

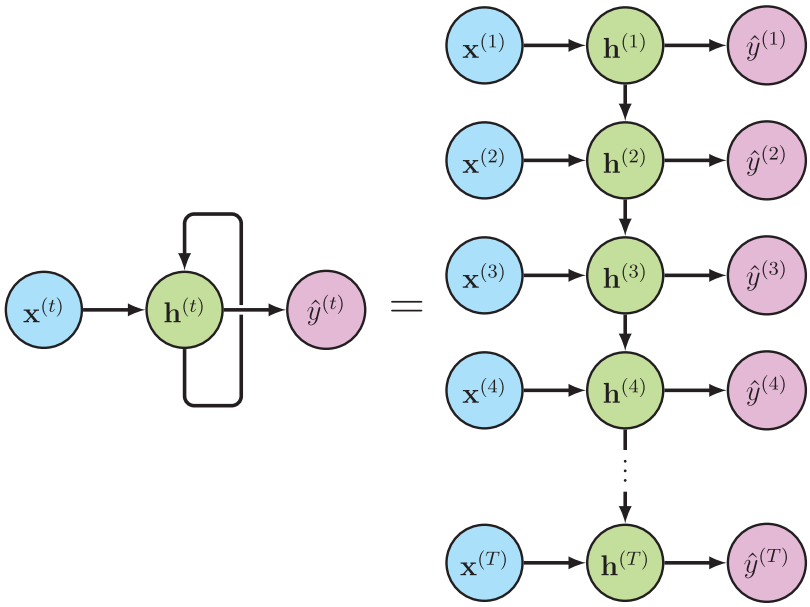
$$\begin{aligned} y^{(1)} &= g\big((\mathbf{b}^{\text{yh}})^\top f(\mathbf{B}^{\text{hx}}\mathbf{x}^{(1)} + \mathbf{b}_{\mathbf{0}}^{\text{hx}}) + b_0^{\text{yh}}\big) + \varepsilon^{(1)}, \\ y^{(2)} &= g\big((\mathbf{b}^{\text{yh}})^\top f(\mathbf{B}^{\text{hx}}\mathbf{x}^{(2)} + \mathbf{B}^{\text{hh}} f(\mathbf{B}^{\text{hx}}\mathbf{x}^{(1)} + \mathbf{b}_{\mathbf{0}}^{\text{hx}}) + \mathbf{b}_{\mathbf{0}}^{\text{hx}}) + b_0^{\text{yh}}\big) \\ &\quad + \varepsilon^{(2)}, \\ &\vdots \\ y^{(T)} &= g\big((\mathbf{b}^{\text{yh}})^\top f(\mathbf{B}^{\text{hx}}\mathbf{x}^{(T)} + \ \ldots\ + \mathbf{B}^{\text{hh}} f(\mathbf{B}^{\text{hx}}\mathbf{x}^{(2)} + \mathbf{B}^{\text{hh}} f(\mathbf{B}^{\text{hx}}\mathbf{x}^{(1)} \\ &\quad + \mathbf{b}_{\mathbf{0}}^{\text{hx}}) + \mathbf{b}_{\mathbf{0}}^{\text{hx}})\ \ldots\ ) + b_0^{\text{yh}}\big) + \ \varepsilon^{(T)}. \end{aligned} \tag{36}$$

Equation 36 is simply Equation 35a substituted into Equation 35 and written down explicitly for all time steps. Schematic representations of Equations 35, 35a, and 36 are presented in Figure 7. The schematic on the left hand side of Figure 7 represents Equations 35 and 35a 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 7 represents the unfolded simple RNN in Equations 36 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 8a). 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. 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 8b). 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 8c). 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.

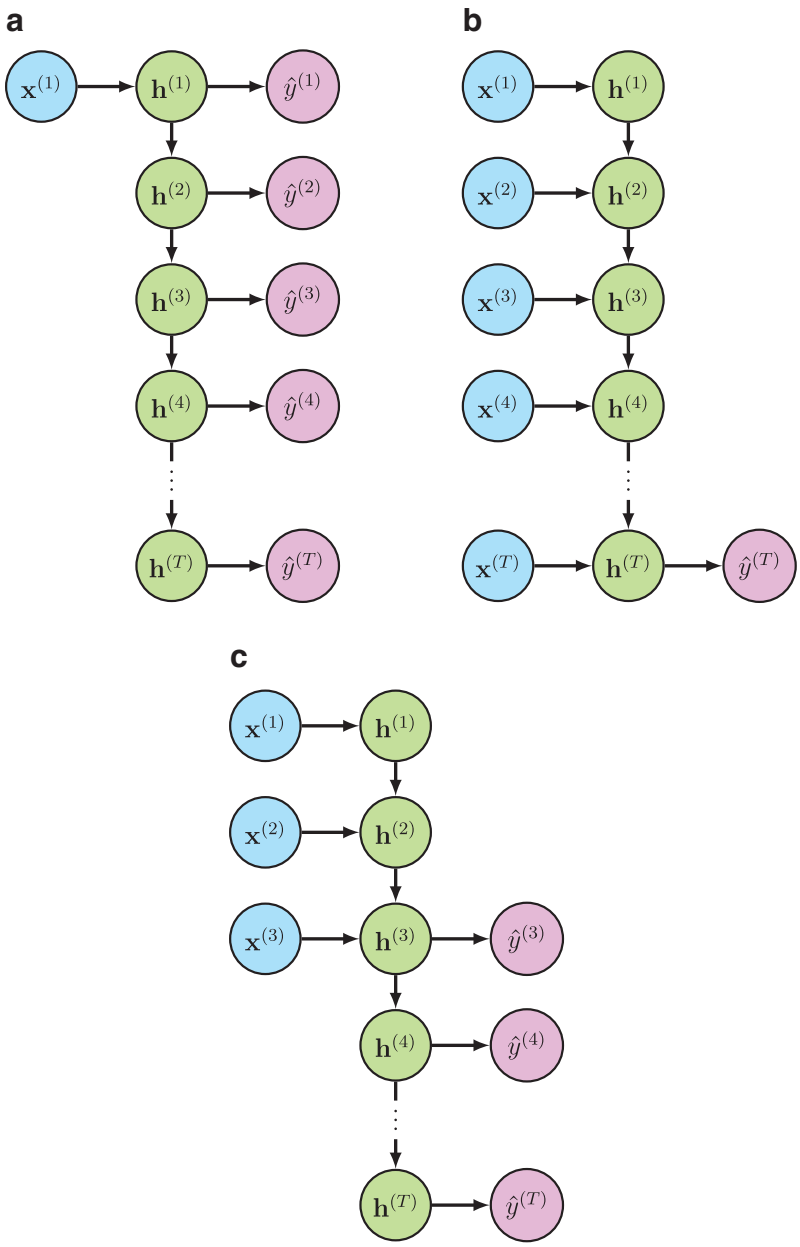
RNNs are appealing in theory because they use information from the past to predict future output values. In practice, however, simple RNNs struggle to use information from very far in the past (Bengio et al., 1994; Doya, 1993; Hochreiter, 1991; Pascanu et al., 2013). This may not be a problem if we only expect recent information to influence the current output value. For example, imagine collecting information about individuals' 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 we might expect their mood yesterday or earlier to only weakly influence their nighttime mood. A simple RNN might be well-

Figure 7
Schematic Representation of the Single-Layer Recurrent Neural Network Both as a Loop and as an Unfolded Loop



Note. See the online article for the color version of this figure.

Figure 8
Schematic Representations of Several Common Recurrent Neural Network Architectures



Note. (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. See the online article for the color version of this figure.

suited 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. Gated RNNs have been effective for some sequence modeling problems including speech recognition, machine translation, and image captioning (Goodfellow et al., 2016). Lipton et al. (2015) provide an accessible overview of gated RNNs that have demonstrated good performance in several practical applications.

By now, some readers may have noticed that RNNs are quite similar to classical time series approaches such as *dynamic factor*

analysis (DFA; e.g., Lütkepohl, 2005; Molenaar, 1985) that model the relationships between latent variables over time. For example, DFA models relations among latent variables at the current time as a function of the latent variables at previous, or *lagged*, time steps. In a conceptually similar manner, RNNs update their hidden layer values at the current time point using hidden layer values from the previous time point and may even explicitly include higher-order lagged relations between the hidden layers (e.g., relations between $\mathbf{h}^{(t-2)}$ and $\mathbf{h}^{(t)}$ and so on). Although they are similar, RNNs and classical approaches like DFA differ in several ways. In particular, RNN-based approaches improve on classical time series approaches in that they automatically account for nonlinear relationships between variables, they do not require conducting data reduction using latent variables, they do not require all individuals to have variability on each variable, and they do not require researchers to prescreen the data for multicollinearity. We note, however, that the benefits of RNNs come at the cost of model interpretability.

In the previous section, we discussed how making FNNs deeper by adding more hidden layers often leads to increased predictive accuracy. Building deep RNNs is less straightforward than building deep FNNs because there are multiple ways to add more hidden layers. In particular, hidden layers can be added to RNNs in three ways: (a) Between the input nodes and the hidden layer nodes at each time point, (b) between hidden layer nodes at subsequent time points, and (c) between the hidden layer nodes and the output nodes at each time point. Each kind of deep RNN may be more or less suited for different kinds of data sets (Pascanu et al., 2014). Determining which kind of deep RNN is best suited for a particular data set is often a matter of experimentation.

We now present a toy example demonstrating the possible application of RNNs to forecasting risk of suicidal ideation (SI; i.e., thinking about or considering suicide; Klonsky et al., 2016) using daily diary data. Imagine that we collect one week of daily diary data for a clinical sample of 1,000 individuals who are at risk for SI. Every day, our individuals receive two cell phone notifications. The first notification arrives in the morning and asks each respondent to rate their current mood using a five-category scale ranging from *extremely good* (1) to *extremely bad* (5). The second notification arrives at night and asks individuals to rate their mood using the same five-category scale as well as to indicate whether or not they thought about committing suicide at any time during the day (*yes* = 1, *no* = 0).

Our goal is to determine whether previous days' moods and SI can be used to forecast whether or not respondents will engage in SI on day seven. We model this problem using a many-to-one RNN wherein respondents' moods and SI on days one through six are the inputs (i.e., $\mathbf{x}^{(t)}$ for $t = 1, \ldots, 6$) and SI on day seven is the output (i.e., $\hat{y}^{(7)}$). Figure 9 steps through how computation proceeds in our RNN approach to forecasting SI.

The key idea underlying these computations is that the RNN's hidden state evolves over time based on current and lagged information. At each time step, say time step t, we start with a vector of daily diary responses $\mathbf{x}^{(t)}$ as well as a hidden state from the previous day $\mathbf{h}^{(t-1)}$. To compute the current hidden state $\mathbf{h}^{(t)}$, we first multiply the current responses $\mathbf{x}^{(t)}$ by the input weight matrix $\mathbf{B}^{\text{hx}}$ and multiply the lagged hidden state $\mathbf{h}^{(t-1)}$ by the recurrent weight matrix $\mathbf{B}^{\text{hh}}$. After adding the resulting vectors together, we add a vector of intercepts $\mathbf{b}_0^{\text{hx}}$ and apply a ReLU activation function f to produce the current hidden state $\mathbf{h}^{(t)}$. At the final time step (i.e., $t = 6$), we use only the final hidden state $\mathbf{h}^{(6)}$ to predict whether or not the individual will engage in SI on day seven. Notice that the hidden state is updated using the same parameters at each time step—that is, RNN parameters are *time-invariant*. Computing model updates this way is similar to the approach used in *vector autoregression* (VAR; Lütkepohl, 2005), a classical time series approach to modeling the relationships between observed variables over time.[4]

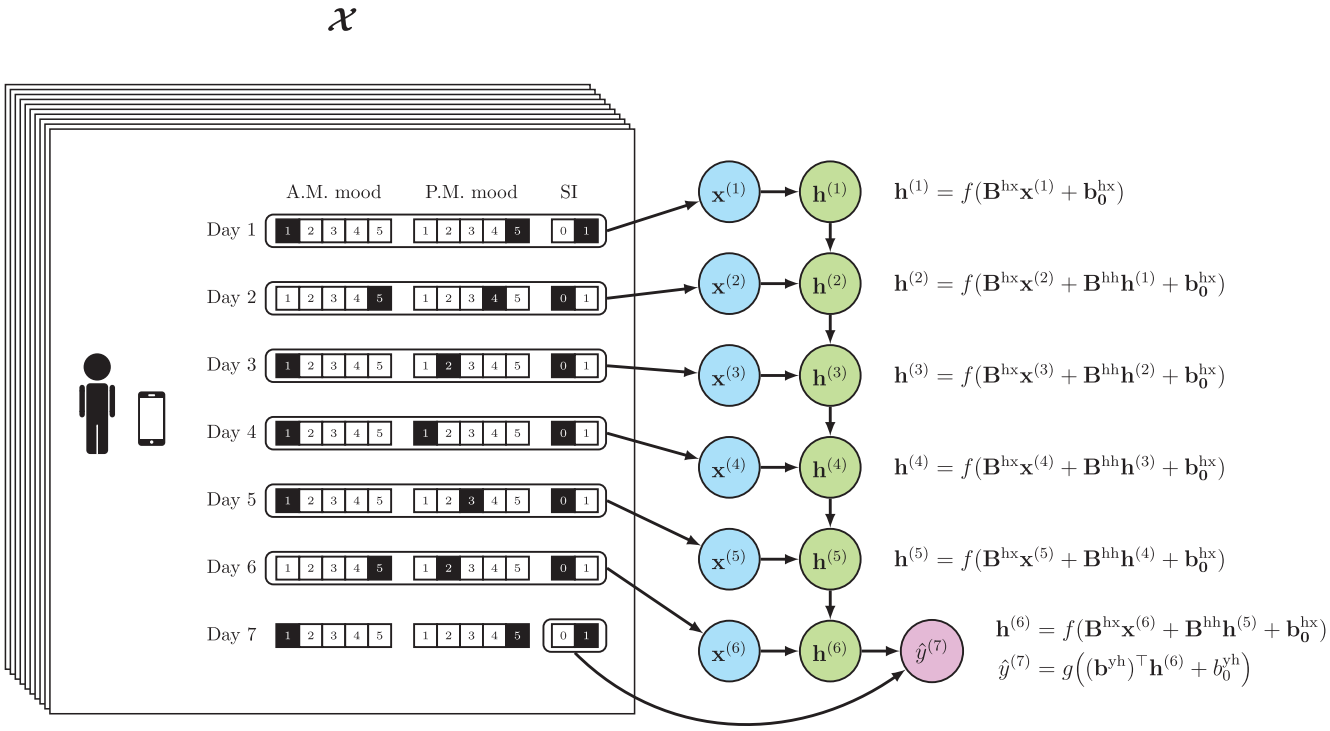
Owing to the close relationship between RNNs and classical time series approaches such as VAR, we used R to simulate data from an ordinal VAR model of lag order six, which is essentially an ordinal regression model (McCullagh, 1980) in which the observed variables at the current time step are regressed on the observed variables at the previous six time points. Creal et al. (2013) provide a full description of *generalized autoregressive score* (GAS) models, of which our ordinal VAR model is a special case where the observed variables at the current time point are multinomially distributed conditional on the previous time point. Our model included three ordinal variables: *Morning mood*, which had five response categories ranging from 1 to 5; *nighttime mood*, which had five categories response categories ranging from 1 to 5; and *suicidal ideation* (SI), which had two response categories of either 0 or 1. To sample sequences from this model, we needed to specify six autoregressive (AR) weight matrices describing the relationships between current and past observed variable values as well as three vectors of strictly ordered category intercepts (i.e., one intercept vector per variable; these are equivalent to the threshold parameters used in ordinal regression).

We set the first-order AR (i.e., AR[1]) weight matrix to the values given in Table 2. We chose response category 1 (i.e., *extremely good* mood) as the reference category for morning and nighttime mood and response category 1 (engaging in SI) as the SI reference category. We therefore only needed to specify AR weights for the nine possible nonreference categories. Choosing the values in Table 2 ensured being in a very bad mood either in the morning or at night was highly predictive of SI and that the effects of previous days' moods on individuals' current moods tended to diminish with increasing time. More specifically, the chosen values were motivated as follows: (a) Having a good morning mood or nighttime mood (response categories 2 or 3) on the previous day should not affect the chances of having a good morning or nighttime mood or engaging in SI on the next day relative to the reference category (zero coefficients on lag 1 morning and nighttime good moods); (b) having a bad morning mood or nighttime mood (response categories 4 or 5) or engaging in SI on the previous day should decrease the chances of having a good morning or nighttime

[4] Readers familiar with VAR may notice that the recurrent weight matrix $\mathbf{B}^{\text{hh}}$ closely corresponds to the time-invariant weight matrix used in a VAR model of order one. The difference between these approaches is that the RNN recurrent weight matrix is used to update an unobserved hidden state, whereas the VAR weight matrix is used to predict observed variable values at each time step.

Figure 9

Schematic Visualizing How Recurrent Neural Networks May Be Applied to Daily Diary Data to Predict Suicidal Ideation



Note. Data for a single simulated participant is shown; the participant's response to a given question at a given time point is shaded in black. SI = suicidal ideation. See the online article for the color version of this figure.

mood on the next day and increase the chances of engaging in SI on the next day (positive coefficients on lag 1 morning and nighttime bad moods and SI). We constructed higher-order AR weight matrices by multiplying the AR(1) weight matrix parameters by the following constants: 0.5 for AR(2), 0.4 for AR(3), 0.3 for AR(4), 0.2 for AR(5), and 0.1 for AR(6). Constructing AR weight matrices this way was motivated by the idea that observations from more distant time points should have less influence on the current observations than more recent observations.

Intercept parameters for both morning and nighttime moods were set to −0.8, −0.2, 0.2, and 0.8, corresponding to the thresholds between response categories 1 and 2, 2 and 3, 3 and 4, and 4 and 5, respectively. The intercept parameter for SI was set to 0.5 so that individuals from our clinical sample would be predisposed to engaging in SI. After specifying our data generating model parameters, we sampled a data set of 1, 000 sequences of length six, discarding the first 100 observations from each sampling run for burn-in. Approximately 21% of simulated individuals engaged in SI on day seven (i.e., had $y^{(7)} = 1$).

We then used the R interface to Keras to fit a gated RNN model called a *long short-term memory* neural network (LSTM; Hochreiter & Schmidhuber, 1997) using 700 simulated sequences, which we evaluated using the remaining 300 sequences. A complete explanation of LSTMs is beyond the scope of this primer but is available in Lipton et al. (2015). We constructed our LSTM as follows:

Table 2

First-Order Autoregressive Weight Matrix Parameters for Simulated Ordinal Vector Autoregression Data

Variable	Response category	Current A.M. mood	Current P.M. mood	Current SI
Lag 1 A.M. mood	2	0	0	0
	3	0	0	0
	4	0.5	0.5	1.0
	5	0.5	0.5	1.0
Lag 1 P.M. mood	2	0	0	0
	3	0	0	0
	4	0.5	0.5	1.0
	5	0.5	0.5	1.0
Lag 1 SI	1	0.5	0.5	2.0

Note. "Current" indicates variable value at current time step, and "Lag 1" indicates variable value at previous time step. SI = suicidal ideation.

```
rnn = keras_model_sequential()
rnn %>%
  layer_lstm(units = 32,
             activation = 'relu,'
             input_shape = c(6, 9)) %>%
  layer_dense(units = 32,
             activation = 'relu') %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 1,
             activation = 'sigmoid')
```

`layer_lstm()` adds an LSTM hidden layer with 32 hidden units and a ReLU activation function to our model. Our input consists of sequences of length six where each observation in the sequence is a nine-dimensional vector of dummy variables, so we set `input_shape = c(6, 9)`. We next add a single FNN layer to our LSTM, then apply a regularization technique called *dropout* using `layer_dropout()` (Srivastava et al., 2014). We complete the model specification with a single output node with a sigmoid activation function.

We used the same model fitting parameters that were used for our single-layer FNN toy example:

```
rnn %>% compile(
  loss = 'binary_crossentropy',
  optimizer = optimizer_adam(),
  metrics = c('accuracy')
)
```

Specifically, we specify the binary cross-entropy objective function, the Adam optimizer, and the accuracy metric for assessing performance. Finally, we fitted and evaluated the model as follows:

```
history = rnn %>% fit(
  train_data$X,
  as.matrix(train_data$y),
  epochs = 300,
  batch_size = 64
)
rnn %>% evaluate(test_data$X,
                 as.matrix(test_data$y))
```

The first input to `fit()` is a 700 × 6 × 9 tensor of training set sequences and the second input is a vector of outcome values. We fit the model for 300 epochs (i.e., passes through the full data set; see **Single-layer FNN toy example: Predicting alcohol use disorder using cross-sectional survey data**) and randomly sample 64 sequences for each fitting iteration. Finally, we tested our LSTM on a test set of 30% of observations. We also calculated sensitivity and specificity using the R package caret (Kuhn, 2008). Code for the full example is available as online supplemental material.

Our fitted LSTM achieved fair performance on the simulated data with relatively little hyperparameter tuning (accuracy = 0.80, sensitivity = 0.30, specificity = 0.87). We suspect that our LSTM struggled to model the low-order autoregressive effects underlying the data as described in Gers et al. (2001). Briefly, Gers et al. (2001) note that whereas classical time series models such as VAR explicitly specify lagged effects, typical RNN implementations rarely specify lagged effects and instead require lagged information to be stored in the model's hidden layer. It may be challenging for RNNs without lagged effects to "learn" when to store and when to overwrite lagged information, although we suspect this problem could be alleviated by greatly increasing the size of the RNN hidden layer or by fitting the RNN for many epochs using adequate regularization. If we wished to obtain better performance on our simulated data using an alternative ANN approach, we could likely instead use a CNN because CNNs are inherently well suited to leverage effects that are close together in time to obtain high predictive accuracy. We now turn to these models.

Convolutional Neural Networks

Just like RNNs are specialized for processing sequential data, *convolutional neural networks* (CNNs) are specialized for processing image data. In both cases, the ordering of the data matters. Just like FNNs and RNNs, 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 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 abstract concepts (e.g., types of objects like faces or shirts) by summarizing less abstract 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 13 a describes how to produce $\mathbf{h}$ in a single-layer FNN: Multiply $\mathbf{x}$ by a weight matrix $\mathbf{B}^{hx}$, add an intercept vector $\mathbf{b}^{hx}$, then apply an element-wise activation function f. 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 $\mathbf{B}^{hx}$ 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.

Before explaining the single-layer CNN, we explain our notation for describing image data sets. In this primer, we focus on data sets containing N different grayscale (i.e., "black-and-white") images. Such data sets may contain fascinating psychological insights (e.g., see **CNN toy example: Predicting emotions from facial expressions**). 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)th number describes the color of the (j, k)th pixel in the image. Since

each observation is a matrix, our data set is no longer a single matrix as in linear regression—rather, it is a collection of matrices. If each image in our data set is the same shape, say $r^{\mathrm{x}} \times c^{\mathrm{x}}$, we can think of our data set as an $N \times r^{\mathrm{x}} \times c^{\mathrm{x}}$ tensor $\boldsymbol{\mathcal{X}}$ where $\mathbf{X}_i$ is a matrix representing the ith image in the data set and $x_{i,\,j,\,k}$ is a number representing the color of the (j, k)th pixel in this image. In Figure 10, we visualize a tensor data set containing four different randomly generated 4×4 grayscale images. We will also imagine that our data set includes an outcome value y_i associated with image i for all $i = 1, \ldots, N$. As usual, we drop all i subscripts from equations—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

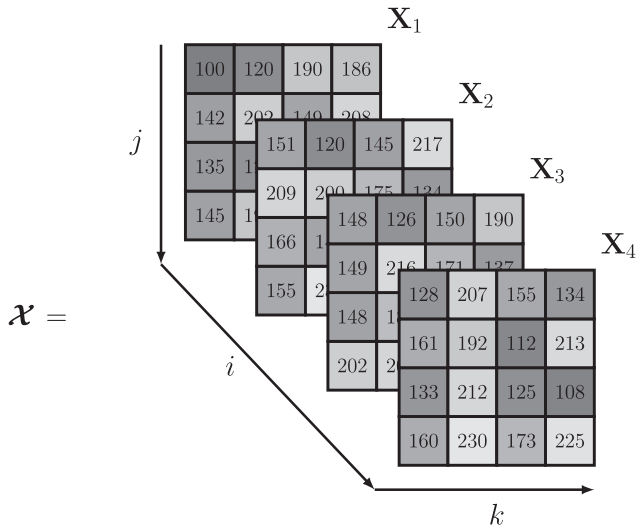
$$s_{j,k} = (\mathbf{X} * \mathbf{B}^{\mathrm{hx}})(j,k) = \sum_{m=j-1}^{j-r^{\mathrm{b}}} \sum_{n=k-1}^{k-c^{\mathrm{b}}} x_{m,n} b^{\mathrm{hx}}_{j-m,k-n},$$

$$j = 1, \ \ldots, r^{\mathrm{x}} + r^{\mathrm{b}} - 1, \ k = 1, \ \ldots, c^{\mathrm{x}} + c^{\mathrm{b}} - 1, \qquad (37)$$

where $*$ is called the *convolution operator*, $\mathbf{X}$ is the $r^{\mathrm{x}} \times c^{\mathrm{x}}$ input image, $\mathbf{B}^{\mathrm{hx}}$ is an $r^{\mathrm{b}} \times c^{\mathrm{b}}$ weight matrix called the *kernel*, and $s_{j,k}$ are elements of the $(r^{\mathrm{x}} + r^{\mathrm{b}} - 1) \times (c^{\mathrm{x}} + c^{\mathrm{b}} - 1)$ matrix $\mathbf{S}$ called the *convolution* of $\mathbf{X}$ and $\mathbf{B}^{\mathrm{hx}}$. Equation 37 looks complicated but is fairly intuitive to understand. "$\mathbf{X} * \mathbf{B}^{\mathrm{hx}}$" is read as "$\mathbf{X}$ convolved with $\mathbf{B}^{\mathrm{hx}}$." Convolving the input image $\mathbf{X}$ with the kernel (i.e., weight matrix) $\mathbf{B}^{\mathrm{hx}}$ produces a new matrix $\mathbf{S}$, much like multiplying two matrices produces a new matrix. The kernel $\mathbf{B}^{\mathrm{hx}}$ is just a small weight 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 11. Note that our figure only visualizes the output values $s_{j,k}$ where the kernel fits completely inside the image, called the *valid convolution*.

Figure 10

Example of an Image Data Set Containing Four 4×4 Random Grayscale Images Formatted as a $4 \times 4 \times 4$ Tensor



We previously mentioned that single-layer CNNs replace multiplying the input image $\mathbf{X}$ by a weight matrix $\mathbf{B}^{\mathrm{hx}}$ with convolving $\mathbf{X}$ and $\mathbf{B}^{\mathrm{hx}}$. 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 f:

$$h_{j,k} = f(b^{\mathrm{hx}}_{j,k,0} + s_{j,k}),$$

$$j = 1, \ \ldots, r^{\mathrm{x}} + r^{\mathrm{b}} - 1, \ k = 1, \ \ldots, c^{\mathrm{x}} + c^{\mathrm{b}} - 1, \qquad (38)$$

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 nonlinear activation function helps single-layer CNNs model outcomes y that vary nonlinearly 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:

$$y = g\left(b^{\mathrm{yh}}_0 + \sum_{j=1}^{r^{\mathrm{h}}c^{\mathrm{h}}} b^{\mathrm{yh}}_j h_j\right) + \varepsilon, \qquad (39)$$

$$\mathbf{h} = \mathrm{vec}(\mathbf{H}), \qquad (39a)$$

where vec, 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 $\mathbf{h}$,[5] h_j are elements of $\mathbf{h}$, and ε is a random error term. As with FNNs, choosing the activation function g determines whether the single-layer CNN will perform regression or classification.

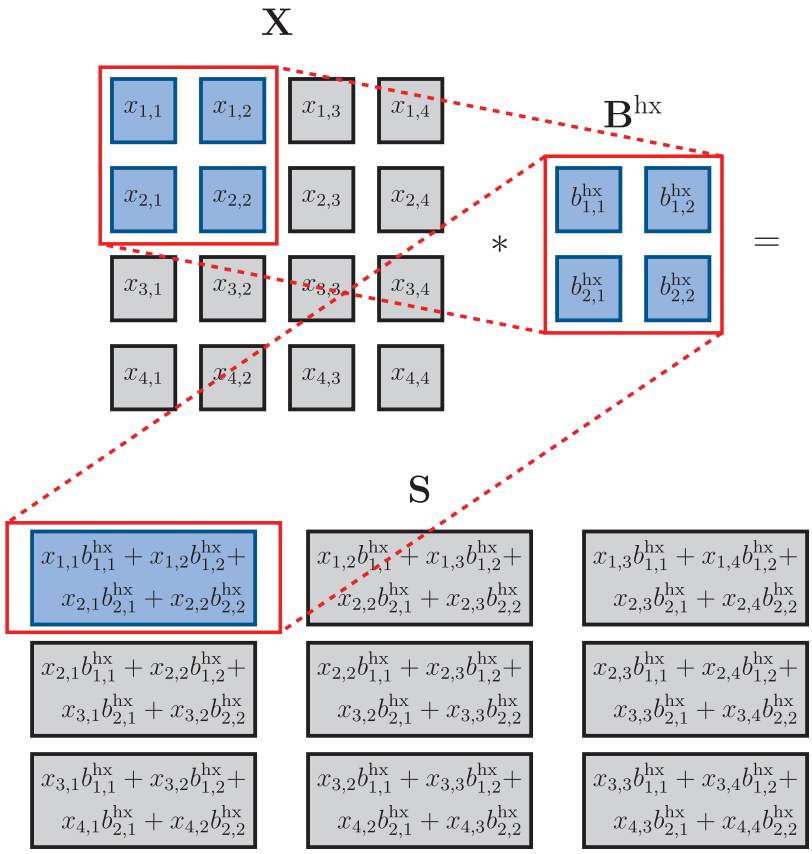
Deep CNNs have revolutionized predictive modeling using image data as well as video, speech, and audio data (LeCun et al., 2015). Basic deep CNNs can be constructed from single-layer CNNs just like deep FNNs can be constructed from single-layer FNNs: Simply add many hidden layers, each feeding in to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

Identifying Important Parts of Images With Two-Dimensional Discrete Convolution

We now concretely demonstrate how the convolution operation may identify important components of images. Figure 12 presents a 3×3 kernel $\mathbf{B}^{\mathrm{hx}}$ that is often used for identifying vertical edges in images. Directly under the kernel, we show the result of applying this kernel to a grayscale image of a face. Notice that in the convolution of the original image and the kernel, vertical edges are visible as white or light gray lines whereas the rest of the image is dark.

[5] 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.

Figure 11
Example of Two-Dimensional Discrete Valid Convolution With a 4 × 4 Image Input and a 2 × 2 Kernel



Note. ∗ denotes the convolution operator. See the online article for the color version of this figure.

At the bottom of Figure 12, we show the result of convolving the kernel with two adjacent 3 × 3 patches of the original image. In the first patch, we can tell that there is a vertical edge between the second and third columns: The first column is dark and has small pixel values, whereas the second column is light and has large pixel values. When we convolve this patch with the kernel, the large pixel values in the first column cause the result to equal to one so that the resulting pixel is essentially black. Note that since grayscale pixel values only range from zero to 255, negative values produced by computing the convolution are treated as zeros and the resulting pixels are set to black.

In the adjacent patch, the vertical edge is now between the first and second columns. The large pixel values in the second column cause the convolution to be a large positive value so the resulting pixel is set to a light gray. Placing the light resulting pixel next to the dark resulting pixel in the convolution of the original image shows us that there was a vertical edge in this patch of the original image. We note that in practice, fitting CNNs will likely result in different fitted kernels than the one presented in Figure 12. Other kernels can identify different kinds of edges (e.g., horizontal or diagonal edges) or even other, less human-interpretable components of images.

CNN Toy Example: Predicting Emotions From Facial Expressions

We now provide a toy example to clarify the application of CNNs to psychological data. The data set used in this example is publicly available on Kaggle, an online community for data scientists and machine learning practitioners (https://www.kaggle.com/shawon10/ckplus). It is a small subset of the Extended Cohn-Kanade data set for facial expression recognition (Kanade et al., 2000; Lucey et al., 2010) and may be similar in size to data sets typically collected by psychologists. As with our FNN example, this example seeks to classify observations into discrete categories.

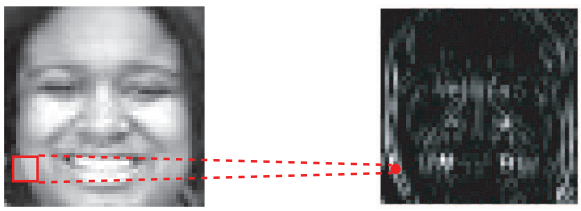
Our data set contains 327 $r^{\text{x}} \times c^{\text{x}} = 48 \times 48$ grayscale images of faces. Each face is labeled with one of the following emotions: *anger* (45 observations), *contempt* (18 observations), *disgust* (59 observations), *fear* (25 observations), *happy* (69 observations), *sad* (28 observations), or *surprise* (83 observations). Each image is duplicated three times in the data set for a total of 981 images used for fitting purposes. Our goal is to use a CNN to predict which

Figure 12

A Demonstration of How Two-Dimensional Discrete Convolution Can Find Vertical Edges in Images

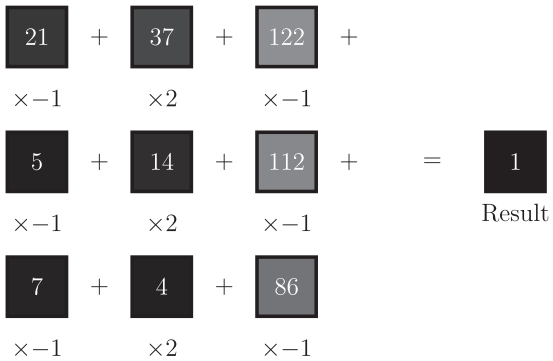
$$\mathbf{B}^{\text{hx}} = \begin{bmatrix} -1 & 2 & -1 \\ -1 & 2 & -1 \\ -1 & 2 & -1 \end{bmatrix}$$

Kernel for vertical edge detection

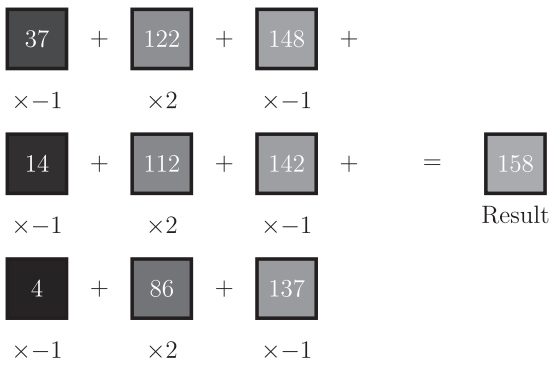


Original image

Convolution



A patch of the original image



An adjacent patch of the original image

Note. Image copyright 2000 by Jeffrey Cohn. See the online article for the color version of this figure.

emotion a face is showing. We might build a model like this to answer any of a variety of psychological research questions. One simple use of such a classification model might be to categorize individuals' reactions to specific stimuli without having to ask them for self-report responses, which poses a potential cognitive burden on an emotional task.

Figure 13 walks through the process of modeling a single image from our data set using a single-layer CNN. We first select an image $\mathbf{X}$ from the tensor data set $\mathcal{X}$. We then convolve our chosen image with a small kernel weight matrix $\mathbf{B}^{\text{hx}}$ to get a new matrix $\mathbf{S}$ (i.e., the convolution). In the figure, we choose the kernel to be the same $r^{\text{b}} \times r^{\text{b}} = 3 \times 3$ kernel used in Figure 12 to detect vertical edges in images. We note that $\mathbf{S}$ is typically smaller than the original image $\mathbf{X}$. In Figure 13, we force the kernel to stay inside of the input image (i.e., we only compute the valid convolution) so that $\mathbf{S}$ has size 46×46.

Once we have the convolution $\mathbf{S}$, we can compute the hidden layer representation $\mathbf{H}$ of the input image by adding an intercept to each element of $\mathbf{S}$ and applying a ReLU activation function. In the figure, we set all intercepts to a large value (i.e., $b^{\text{hx}}_{j,k,0} = 30$ for $j, k = 1, \ldots, 48$) and applied the ReLU function to demonstrate the impact of this processing step. The resulting image $\mathbf{H}$ is a modified version of the convolution $\mathbf{S}$ in which only the most salient vertical edges are now visible. Identifying salient vertical edges as shown might be useful for our toy CNN, which could, for example, utilize the lines representing the individual's teeth to predict that they are happy. Notice that $\mathbf{H}$ is still square-shaped. To predict our emotion outcome, we first flatten $\mathbf{H}$ into a vector $\mathbf{h}$. We then multiply $\mathbf{h}$ by a weight vector, add an intercept, and apply a softmax activation function (Equation 12). The softmax function lets our model output seven probabilities, one corresponding to each possible emotion the face could be showing. The highest predicted probability corresponds to the emotion that the CNN most strongly believes the face is showing.

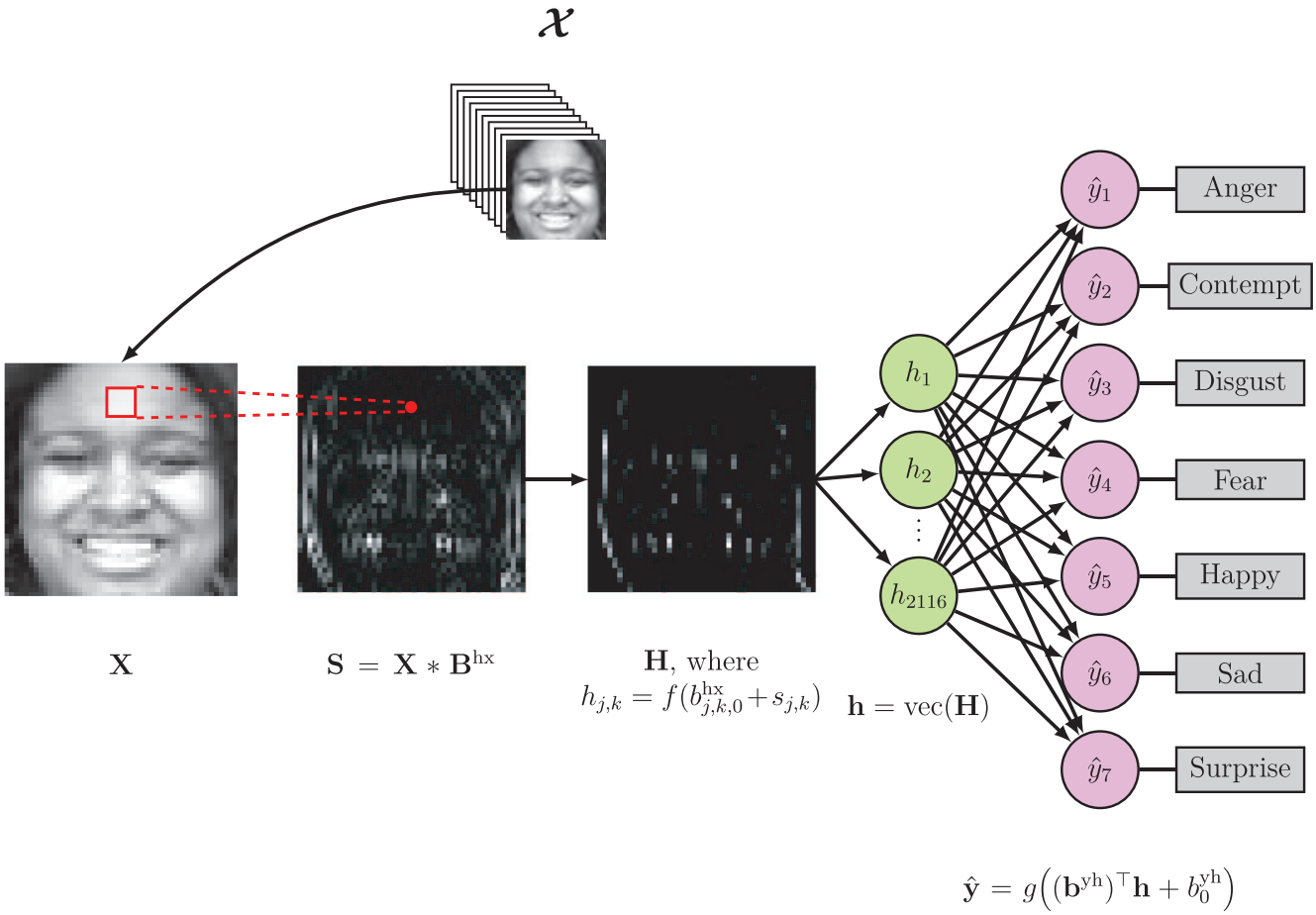
In practice, CNNs often require many convolutional layers composed together to obtain high predictive accuracy. Although complete descriptions of state-of-the-art CNN techniques are beyond the scope of this primer, we demonstrate the feasibility of this type of analysis by fitting a deep CNN to our facial emotion data set using the R interface to Keras. Complete descriptions of the techniques used below are beyond the scope of this primer but are available in Goodfellow et al. (2016).

As with our other ANN models, we constructed our deep CNN one layer at a time:

```
cnn = keras_model_sequential()
cnn %>%
  layer_conv_2d(filters = 6,
                kernel_size = c(5, 5),
                input_shape = c(48, 48, 1),
                padding = 'same',
                activation = 'relu') %>%
  layer_max_pooling_2d(
      pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 16,
                kernel_size = c(5, 5),
                padding = 'same',
                activation = 'relu') %>%
  layer_max_pooling_2d(
      pool_size = c(2, 2)) %>%
  layer_conv_2d(filters = 64,
                kernel_size = c(3, 3),
                padding = 'same',
                activation = 'relu') %>%
  layer_max_pooling_2d(
```

Figure 13

Schematic Visualizing How Single-Layer Convolutional Neural Networks May Be Applied to Images of Faces to Predict Emotions



Note. Image copyright 2000 by Jeffrey Cohn. See the online article for the color version of this figure.

```
    pool_size = c(2, 2)) %>%
  layer_flatten() %>%
  layer_dense(units = 128,
              activation = 'relu') %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 7,
              activation = 'softmax')
```

`layer_conv_2d()` performs two-dimensional discrete convolution, adds intercepts, and applies an activation function to its input. Our input consists of 48 × 48 grayscale images, so we set `input_shape = c(48, 48, 1)`. The final value of 1 tells the model that we are working with grayscale images; if we had colored images, we would set this value to `3`. We choose a 5 × 5 kernel by setting `kernel_size = c(5, 5)` and choose a ReLU activation function. Setting `padding = 'same'` applies a full convolution (i.e., Equation 37) instead of the valid convolution described previously. By setting `filters = 6`, we choose to fit six different kernels rather than a single kernel. Each fitted kernel may learn to identify different components of the input images, such as different kinds of edges or lines. This corresponds to including six nodes in the hidden layer of a single-layer FNN.

We applied a *max pooling* layer after our convolutional layer using `layer_max_pooling_2d()`. Max pooling layers are used in CNNs to condense information as it passes through the model. Choosing `pool_size = c(2, 2)` decreases both the height and width of the convolved image by a factor two. Goodfellow et al. (2016) provide a full description of max pooling.

After applying two more convolutional and max pooling layers in succession, we flatten our convolved images into a vector using `layer_flatten()`. We apply a single-layer FNN to the flattened vector using `layer_dense()`, then apply the dropout regularization technique using `layer_dropout()` (Srivastava et al., 2014). The final `layer_dense()` outputs of vector of seven probabilities (i.e., one probability for each possible emotion) using the softmax activation function (Equation 12).

Next, we specified parameters for model fitting:

```
cnn %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_adam(),
  metrics = c('accuracy')
)
```

This specification is similar to the one given for our single-layer FNN and RNN toy examples. However, we now set `loss = 'categorical_crossentropy'`, which extends the binary cross-entropy objective function to outcomes with more than two categories. More formally, the categorical cross-entropy is the log-likelihood of an outcome that follows a multinomial distribution with number of cells equal to the number of outcome categories, cell probabilities predicted by the fitted model, and trial size one. It is the same objective function used for fitting ordinal regression models (Kutner et al., 2004).

We fitted and tested the model:

```
history = cnn %>% fit(
  train_data$X,
  as.matrix(train_data$y),
  epochs = 50,
  batch_size = 7
)
cnn %>% evaluate(test_data$X,
                 as.matrix(test_data$y))
```

The first input to `fit()` is a $683 \times 48 \times 48 \times 1$ tensor of training set images, and the second input is an outcome vector. We fitted the model for 50 epochs and randomly sampled seven images for each fitting iteration. Finally, we evaluated our deep CNN on a test set of 30% of observations.

We include a full R script for our analysis as online supplemental material. Our fitted model obtained a predictive accuracy of 0.84 on a test set of 30% of observations, suggesting that our images of faces contain objective information that can be used to predict their emotional content. We could possibly increase our model's predictive accuracy even further either by collecting more data (which could be expensive) or by fitting our model to one of many publicly available large facial expression data sets (e.g., facial expression databases can be found at https://web.archive.org/web/20180325205102/http://emotion-research.net/wiki/Databases or https://www.ecse.rpi.edu/~cvrl/database/other_facial_expression.htm) then refining the fitted parameters on our smaller data set using a transfer learning approach.

Discussion and Future Directions

Deep learning is a successful machine learning paradigm that has revolutionized the psychology-related fields of computer vision and natural language processing. Psychologists will likely find that state-of-the-art deep learning algorithms outperform other machine learning algorithms such as random forests (RFs) and support vector machines (SVMs) when used to predict outcomes of interest in very large data sets with many weakly correlated variables or with sequence (e.g., text, video) or image observations. Although such data sets are not common in mainstream psychological research, they are readily available through web-based and clinical sources. Additionally, psychologists with many other kinds of data sets may also benefit from deep learning: Researchers who utilize transfer learning may reap the benefits of deep learning using small data sets, survey researchers may need to spend less time developing measures when using deep learning to predict outcomes of interest, and researchers with hierarchically structured data (e.g., educational or clinical psychologists) may boost predictive accuracy using multitask deep learning approaches.

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 other machine learning models such as RFs and SVMs, which suggests 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 Global Positioning System (GPS) data will produce large data sets that deep learning models can utilize to accurately predict important psychological outcomes. We are particularly excited by the potential of deep multitask and sequence modeling approaches, which may lead to the development of personalized models for accurately detecting risk (e.g., for suicidal ideation) at the individual level in real time.

In this primer, we have introduced the feedforward neural network (FNN), the recurrent neural network (RNN), and the convolutional neural network (CNN) 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 further 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 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 et al., 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.

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*(1), 104–117. https://doi.org/10.1016/j.cviu.2018.05.001

Arik, S. O., & Pfister, T. (2019). *TabNet: Attentive interpretable tabular learning*. Retrieved from http://arxiv.org/abs/1908.07442

Bai, S., Kolter, J. Z., & Koltun, V. (2018). *An empirical evaluation of generic convolutional and recurrent networks for sequence modeling*. Retrieved from http://arxiv.org/abs/1803.01271

Bakker, B., & Heskes, T. (2004). Task clustering and gating for bayesian multitask learning. *Journal of Machine Learning Research*, *4*(1), 83–99. https://doi.org/10.1162/153244304322765658

Baldi, P. F., & Hornik, K. (1995). Learning in linear neural networks: A survey. *IEEE Transactions on Neural Networks*, *6*(4), 837–858. https://doi.org/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. https://doi.org/10.3758/s13428-016-0787-0

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

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.

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

Bollen, K. A. (2001). Indicator: Methodology. In N. J. Smelser & B. Baltes (Eds.), *International encyclopedia of the social and behavioral sciences*. Elsevier.

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* (pp. 144–152). ACM. https://doi.org/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. https://doi.org/10.1177/1745691610393980

Canzian, L., & Musolesi, M. (2015). Trajectories of depression: Unobtrusive monitoring of depressive states by means of smartphone mobility traces analysis. *Proceedings of the 2015 Association for Computing Machinery International Joint Conference on Pervasive and Ubiquitous Computing* (pp. 1293–1304). ACM. https://doi.org/10.1145/2750858.2805845

Caruana, R. (1997). Multitask learning. *Machine Learning*, *28*(1), 41–75.

Coppersmith, G., Harman, C., & Dredze, M. (2014). Measuring post traumatic stress disorder in twitter. *Proceedings of the 8th International Conference on Weblogs and Social Media* (pp. 579–582). AAAI.

Coppersmith, G., Ngo, K., Leary, R., & Wood, A. (2016). *Exploratory analysis of social media prior to a suicide attempt* (pp. 106–117). https://doi.org/10.18653/v1/w16-0311

Creal, D., Koopman, S. J., & Lucas, A. (2013). Generalized autoregressive score models with applications. *Journal of Applied Econometrics*, *28*(5), 777–795. https://doi.org/10.1002/jae.1279

Cśaji, B. (2001). *Approximation with artificial neural networks* (M.Sc. thesis). https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.101.2647&rep=rep1&type=pdf

De Choudhury, M., Kiciman, E., Dredze, M., Coppersmith, G., & Mrinal, K. (2016). Discovering shifts to suicide ideation from mental health content in social media. *Proceedings of the SIGCHI conference on human factors in computing systems* (pp. 2098–2110). ACM.

De Veaux, R. D., & Ungar, L. H. (1994). Multicollinearity: A tale of two nonparametric regressions. In P. Cheeseman & P. R. W. Olford (Eds.), *Selecting models from data* (pp. 393–402). Springer. https://doi.org/10.1007/978-1-4612-2660-4_40

Doya, K. (1993). Bifurcations of recurrent neural networks in gradient descent learning. *IEEE Transactions on Neural Networks*, *1*, 75–80. 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*(1), 91–118. https://doi.org/10.1146/annurev-clinpsy-032816-045037

Falbel, D., Allaire, J., Chollet, F., Tang, Y., Van Der Bijl, W., Studer, M., & Keydana, S. (2019). R interface to Keras. *GitHub*. Retrieved from https://github.com/rstudio/keras

Friedman, J., Hastie, T., & Tibshirani, R. (2010). Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, *33*(1), 1–22. https://doi.org/10.18637/jss.v033.i01

Gers, F. A., Eck, D., & Schmidhuber, J. (2001). Applying LSTM to time series predictable through time-window approaches. In G. Dorffner, H. Bischof, & K. Hornik (Eds.), *International conference on artificial neural networks* (pp. 669–676). Springer. https://doi.org/10.1007/3-540-44668-0

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier neural networks. *Proceedings of Machine Learning Research*, *15*, 315–323. http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf

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

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

Gorban, A. N., Kégl, B., Wunsch, D. C., & Zinovyev, A. (2008). *Principal manifolds for data visualization and dimension reduction* (Vol. 58). Springer-Verlag.

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. https://doi.org/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. https://doi.org/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. https://doi.org/10.1177/0963721416666518

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: Data mining, inference, and prediction* (2nd ed.). Springer.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. *Proceedings of the IEEE International Conference on Computer Vision* (pp. 1026–1034). https://doi.org/10.1109/ICCV.2015.123

Heaton, J. (2008). *Introduction to neural networks for Java* (2nd ed., Vol. *99*). Heaton Research.

Hochreiter, S. (1991). *Untersuchungen zu dynamischen neuronalen Netzen* [Investigations into dynamic neural networks] (Doctoral dissertation). Technische Universität München. http://www.bioinf.jku.at/publications/older/3804.pdf

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 1735–1780. http://www7.informatik.tu-muenchen.de/hochreit%0Ahttp://www.idsia.ch/juergen

Hotelling, H. (1957). The relations of the newer multivariate statistical methods to factor analysis. *The British Journal of Statistical Psychology*, *10*(2), 69–79.

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* (pp. 157–166). IEEE. https://doi.org/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. https://doi.org/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* (pp. 2146–2153). IEEE.

Kanade, T., Cohn, J. F., & Tian, Y. (2000). Comprehensive database for facial expression analysis. *Proceedings of the 4th IEEE international conference on automatic face and gesture recognition* (pp. 46–53). IEEE. https://doi.org/10.1109/AFGR.2000.840611

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. https://doi.org/10.1016/j.patrec.2014.01.008

Kendall, M. G. (1957). *A course in multivariate analysis*. Griffin.

Kingma, D. P., & Ba, J. L. (2015). *Adam: A method for stochastic optimization*. https://arxiv.org/pdf/1412.6980.pdf

Klonsky, E. D., May, A. M., & Saffer, B. Y. (2016). Suicide, suicide attempts, and suicidal ideation. *Annual Review of Clinical Psychology*, *12*(1), 307–330. https://doi.org/10.1146/annurev-clinpsy-021815-093204

Krämer, N., Boulesteix, A. L., & Tutz, G. (2008). Penalized partial least squares with applications to B-spline transformations and functional data. *Chemometrics and Intelligent Laboratory Systems*, *94*(1), 60–69. https://doi.org/10.1016/j.chemolab.2008.06.009

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burge, L. Bottou, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems* (pp. 1097–1105). NIPS.

Kuhn, M. (2008). Building predictive models in R using the caret package. *Journal of Statistical Software*, *28*(5), 1–26. https://doi.org/10.18637/jss.v028.i05

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. https://doi.org/10.1037/met0000081

Lavalleé, P., & Beaumont, J.-F. (2015). Why we should put some weight on weights. *Survey methods: Insights from the field*. Retrieved from https://surveyinsights.org/?p=6255

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

LiKamWa, R. (2012). *MoodScope: Building a mood sensor from smartphone usage patterns* (PhD thesis). Rice University.

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. https://doi.org/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*. 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*. Retrieved from http://arxiv.org/abs/1709.02540

Lucey, P., Cohn, J. F., Kanade, T., Saragih, J., Ambadar, Z., & Matthews, I. (2010). The extended Cohn-Kanade dataset (CK+): A complete dataset for action unit and emotion-specified expression. *2010 IEEE computer society conference on computer vision and pattern recognition-workshops* (pp. 94–101). https://doi.org/10.1109/CVPRW.2010.5543262

Lütkepohl, H. (2005). *New introduction to multiple time series analysis*. Springer.

McClelland, J. L., Rumelhart, D. E., & P. D. P. Research Group. (1986). *Parallel distributed processing* (Vol. 1). MIT Press.

McCullagh, P. (1980). Regression models for ordinal data. *Journal of the Royal Statistical Society*, *42*(2), 109–142.

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

McNeish, D. M. (2015). Using lasso for predictor selection and to assuage overfitting: A method long overlooked in behavioral sciences. *Multivariate Behavioral Research*, *50*(5), 471–484. https://doi.org/10.1080/00273171.2015.1036965

Meehl, P. E. (1990). Why summaries of research on psychological theories are often uninterpretable. *Psychological Reports*, *66*(1), 195–244. https://doi.org/10.4324/9780203052341

Mehrotra, A., Hendley, R., & Musolesi, M. (2016). Towards multi-modal anticipatory monitoring of depressive states through the analysis of human-smartphone interaction. *Proceedings of the 2016 association for computing machinery international joint conference on pervasive and ubiquitous computing* (pp. 1132–1138). ACM. https://doi.org/10.1145/2968219.2968299

Mehrotra, A., Tsapeli, F., Hendley, R., & Mirco, M. (2017). MyTraces: Investigating correlation and causation between users' emotional states and mobile phone interaction. *Proceeding of the Association for Computing Machinery on Interactive, Mobile, Wearable and Ubiquitous Technologies*, *1*(3), 1281–1284.

Mehta, Y., Majumder, N., Gelbukh, A., & Cambria, E. (2020). Recent trends in deep learning based personality detection. *Artificial Intelligence Review*, *53*(4), 2313–2339. https://doi.org/10.1007/s10462-019-09770-z

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

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

Mitchell, M., Hollingshead, K., & Coppersmith, G. (2015). Quantifying the language of schizophrenia in social media. *Proceedings of the 2nd workshop on computational linguistics and clinical psychology: From linguistic signal to clinical reality* (pp. 11–20). ACL. https://doi.org/10.3115/v1/w15-1202

Molenaar, P. C. (1985). A dynamic factor model for the analysis of multivariate time series. *Psychometrika*, *50*(2), 181–202. https://doi.org/10.1007/BF02294246

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

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted Boltzmann machines. *Proceedings of the 27th international conference on machine learning*. ACM. https://icml.cc/Conferences/2010/papers/432.pdf

Olson, M., Wyner, A. J., & Berk, R. (2018). Modern neural networks generalize on small data sets. *Advances in neural information processing systems* (pp. 3619–3628). ACM.

Pan, S. J., & Yang, Q. (2010). A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, *22*(10), 1345–1359. https://doi.org/10.1109/TKDE.2009.191

Pascanu, R., Gulcehre, C., Cho, K., & Bengio, Y. (2014). How to construct deep recurrent neural networks. *Proceedings of the second international conference on learning representations* (pp. 1–13). https://nyuscholars.nyu.edu/en/publications/how-to-construct-deep-recurrent-neural-networks-proceedings-of-th

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

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. https://doi.org/10.1038/mp.2011.182

Rachuri, K. K., Musolesi, M., Mascolo, C., Rentfrow, P. J., Longworth, C., & Aucinas, A. (2010). EmotionSense: A mobile phones based adaptive platform for experimental social psychology research. *Proceedings of the 2010 association for computing machinery conference on ubiquitous computing* (pp. 281–290). ACM. https://doi.org/10.1145/1864349.1864393

Raudenbush, S. W., & Bryk, A. S. (2002). *Hierarchical linear models: Applications and data analysis methods*. Sage.

R Core Team. (2020). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing. https://www.r-project.org/

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*(6), 386–408. https://doi.org/10.1037/h0042519

Saeb, S., Zhang, M., Karr, C. J., Schueller, S. M., Corden, M. E., Kording, K. P., & Mohr, D. C. (2015). Mobile phone sensor correlates of depressive symptom severity in daily-life behavior: An exploratory study. *Journal of Medical Internet Research*, *17*(7), 1–11. https://doi.org/10.2196/jmir.4273

Schwartz, H. A., Eichstaedt, J., Kern, M. L., Park, G., Sap, M., Stillwell, D., Kosinski, M., & Ungar, L. (2014). Towards assessing changes in degree of depression through Facebook. *Workshop on computational linguistics and clinical psychology: From linguistic signal to clinical reality* (pp. 118–125). ACL. https://doi.org/10.3115/v1/w14-3214

Sezer, O. B., Gudelek, M. U., & Ozbayoglu, A. M. (2020). Financial time series forecasting with deep learning: A systematic literature review: 2005–2019. *Applied Soft Computing Journal*, *90*(1), 106181. https://doi.org/10.1016/j.asoc.2020.106181

Smith, L. N. (2018). *A disciplined approach to neural network hyper-parameters: Part 1 - learning rate, batch size, momentum, and weight decay* (Technical report). U.S. Naval Research Laboratory. http://arxiv.org/abs/1803.09820

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

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*(1), 1929–1958.

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* (pp. 715–724). https://doi.org/10.1145/3038912.3052676

Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., & Liu, C. (2018). *A survey on deep transfer learning*. Springer.

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

Torrey, L., & Shavlik, J. (2009). Transfer learning. In E. Soria, J. Martin, R. Magdalena, M. Martinex, & A. Serrano (Eds.), *Handbook of research on machine learning applications* (pp. 242–264). IGI Global.

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

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems* (pp. 5999–6009). NIPS.

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*(1), 58–75. https://doi.org/10.1016/j.neubiorev.2017.01.002

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

Wold, H. (1966). Nonlinear estimation by iterative least squares procedure. In F. N. David & J. Neyman (Eds.), *Research papers in statistics* (pp. 411–444). Wiley.

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. https://doi.org/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. https://doi.org/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. https://doi.org/10.1177/1745691617693393

Zhang, Y., & Yang, Q. (2017). A survey on multi-task learning. *arXiv*. Retrieved from http://arxiv.org/abs/1707.08114

Zheng, A., & Casari, A. (2018). *Feature engineering for machine learning: Principles and techniques for data scientists*. O'Reilly Media. Retrieved from https://www.amazon.com/Feature-Engineering-Machine-Learning-Principles/dp/1491953241

Appendix A

A Brief Review of Linear Regression

This appendix reviews the linear regression algorithm. Familiarity with the material in this appendix is a prerequisite to understanding the ANN models described in this article.

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

$$y_i = b_0 + \sum_{j=1}^{p} b_j x_{i,j} + \varepsilon_i, \;\; i = 1, \;\; \ldots \;\; , N, \tag{A.1}$$

where y_i is the ith observed value of the outcome variable, $x_{i,j}$ is the ith observed value of the jth predictor variable, b_j is the jth weight parameter, b_0 is the intercept, and ε_i is the error or randomness in y_i that is unaccounted for by the predictors. The intercept b_0 is the predicted outcome value when all of the predictor values are equal to zero.

It is often useful to write Equation A.1 concisely using matrices. To do so, we collect our observed predictors in an $N \times p$ matrix, 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 observed outcomes in an $N \times 1$ vector $\mathbf{y}$. Equation A.1 can then be written as

$$\mathbf{y} = \mathbf{Xb} + \boldsymbol{\varepsilon}, \tag{A.2}$$

where $\mathbf{b}$ is a $(p + 1) \times 1$ vector of weight parameters including the intercept and $\boldsymbol{\varepsilon}$ is an $N \times 1$ vector of errors.

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

$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(y_i - \left(b_0 + \sum_{j=1}^{p} b_j x_{i,j}\right)\right)^2. \tag{A.3}$$

Equation A.3 is the average of the squared differences between the observed outcome values y_i and the predicted outcome values $\hat{y}_i = y_i - \varepsilon_i$. MSE always outputs a positive real number which should be close to zero when the observed and predicted outcome values are very similar and should be large when the observed and predicted outcome values are very different. Equation A.3 is expressed in matrix form as

$$MSE = \frac{1}{N}\|\mathbf{y} - \mathbf{Xb}\|_2^2. \tag{A.4}$$

In Equation A.4, $\|\cdot\|_2$ 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 $\mathbf{x}$, $\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^{N} x_i^2}$).[6]

Finally, we optimize our linear regression model by finding the weight parameters that make our objective 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 to the weight parameters to zero and solving for the weight parameters:

$$\nabla_{\mathbf{b}} \frac{1}{N}\|\mathbf{y} - \mathbf{Xb}\|_2^2 = 0 \Rightarrow \tag{A.5}$$

$$\hat{\mathbf{b}} = (\mathbf{X}^\top \mathbf{X})^{-1}\mathbf{X}^\top \mathbf{y}, \tag{A.6}$$

where $\nabla_{\mathbf{b}}$ denotes the gradient with respect to the weight parameters. Psychologists may recognize Equation A.6, which lets us directly compute the weight parameters $\hat{\mathbf{b}}$ that minimize the MSE.

[6] 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 A.4, 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 outcome values and the predicted outcome values.

(*Appendices continue*)

Appendix B

Machine Learning Approaches to Preventing Overfitting

Statistics and machine learning algorithms are built from the same basic ingredients. For example, the linear regression model and fitting procedure (see Appendix A) are the same in both the statistics and machine learning frameworks. Unlike statistical methods, however, machine learning methods explicitly attempt to prevent overfitting. In this appendix, we first briefly discuss how the goals of classical statistics and machine learning differ. We then explain basic machine learning approaches to preventing overfitting.

Statistics is primarily concerned with *inference*—that is, it aims to make conclusions about the larger population of subjects from which the data set was sampled. This is usually achieved by estimating either p values for or confidence intervals around the model parameters. The overarching goal is to find the true population parameters that generated the data set and to interpret these parameters.

Machine learning is primarily concerned with *generalization*—that is, machine learning aims to build models that perform well with new, previously unseen data sets (e.g., Hastie et al., 2009). 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 fit our model using only the training set.[7] The final value of the objective 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 objective function on a new input observation (Hastie et al., 2009). We can estimate the generalization error by using our fitted model to compute the value of the objective 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 fit the model. A model suffers from *overfitting* when the difference 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.

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 predictors and the outcomes. There are many ways to change the representational capacity of a deep learning model, some of which are discussed throughout the Overview of Artificial Neural Network Models 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 predictor x and one outcome y. If we include an intercept, our model is $y_i = b_0 + b_1x_i + \varepsilon_i$, $i = 1, \ldots, N$; our model parameters are b_0 and b_1; and our objective function is

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - (b_0 + b_1x_i))^2.$$

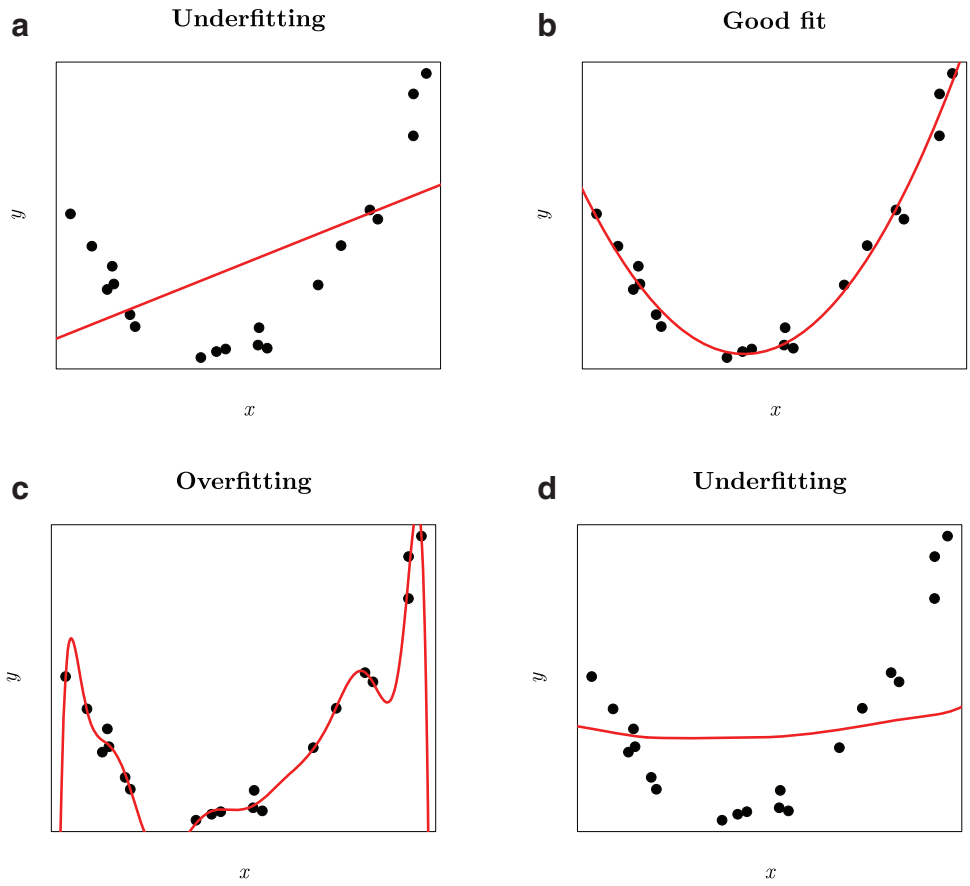
Our hypothesis space is $\mathcal{H} = \{f(x) = b_0 + b_1x : b_0, b_1 \in \mathbb{R}\}$, the set of all two-dimensional lines with real-valued parameters. Our model can therefore approximate any two-dimensional line imaginable. If the data set was generated by a quadratic function like $f(x) = x^2$, however, this model will be unable to approximate the data generating function and will demonstrate underfitting (e.g., Figure B1a). We can increase our model's representational capacity by changing the model to $y_i = b_0 + b_1x_i + b_2x_i^2 + \varepsilon_i$, 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 B1b).

[7] 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.

(*Appendices continue*)

Figure B1

Illustration of the Effects of Altering the Capacity of the Linear Regression Algorithm on Model Fit



Note. (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] penalty. See the online article for the color version of this figure.

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 B1c, a linear regression model with 12 polynomial terms demonstrates overfitting when the data-generating function is quadratic.

(*Appendices continue*)

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 B1c. To reduce overfitting, we can regularize our model by adding a penalty term called a *regularizer* to the objective function:

$$MSE + \underbrace{\lambda \|\theta\|_2^2}_{\text{regularizer}}, \lambda \geq 0, \tag{B.1}$$

where λ is a positive real number called a *regularization parameter*. Modifying an objective 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 objective 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 B1d, regularization eliminates overfitting but reintroduces underfitting.

Hyperparameter Tuning

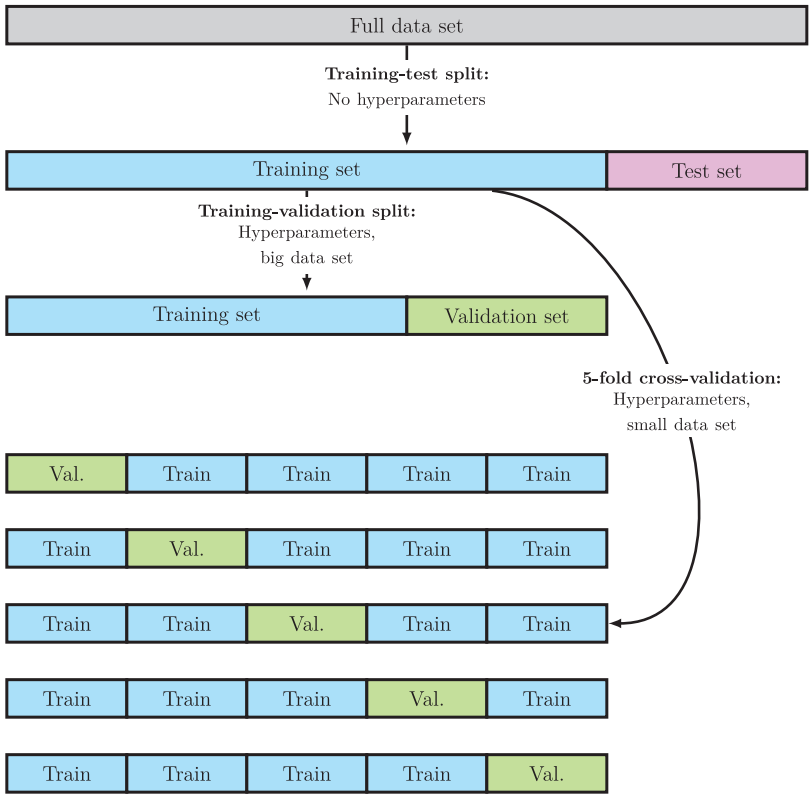
Many machine learning algorithms have settings called *hyperparameters* that affect the behavior of the model but cannot be chosen by the algorithm itself during training. In our linear regression example, the number of polynomial terms included in our model and the value of the λ term were both hyperparameters. 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 fit 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*. 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, the validation set approach is preferred when building deep learning models. Researchers typically only test a few reasonable hyperparameter values until their models achieve adequate accuracy on the validation set.

A cross-validation approach can also be used for hyperparameter tuning, especially when there is not much data available. In *k-fold cross-validation*, we divide the training observations into k non-overlapping groups called *folds*. For i from 1 to k, we test the model on fold i and fit it on the other folds. We then average the test error across all k trials and pick the hyperparameter values that gave the smallest average test error. We illustrate hyperparameter tuning via validation set and 5-fold cross-validation approaches in Figure B2. k-fold cross-validation is an important machine learning tool but is rarely used in deep learning because repeatedly fitting a large number of deep learning models can be very computationally expensive.

(*Appendices continue*)

Figure B2

Schematic Illustrating the Validation Set and Fivefold Cross-Validation Approaches to Hyperparameter Tuning



Note. See the online article for the color version of this figure.

Received September 4, 2019
Revision received July 24, 2020
Accepted September 14, 2020 ■