PSYCHOMETRIKA—VOL. 83, NO. 4, 847–857
DECEMBER 2018
https://doi.org/10.1007/s11336-018-9608-y

AUTOMATED ITEM GENERATION WITH RECURRENT NEURAL NETWORKS

MATTHIAS VON DAVIER
NATIONAL BOARD OF MEDICAL EXAMINERS

Utilizing technology for automated item generation is not a new idea. However, test items used in commercial testing programs or in research are still predominantly written by humans, in most cases by content experts or professional item writers. Human experts are a limited resource and testing agencies incur high costs in the process of continuous renewal of item banks to sustain testing programs. Using algorithms instead holds the promise of providing unlimited resources for this crucial part of assessment development. The approach presented here deviates in several ways from previous attempts to solve this problem. In the past, automatic item generation relied either on generating clones of narrowly defined item types such as those found in language free intelligence tests (e.g., Raven's progressive matrices) or on an extensive analysis of task components and derivation of schemata to produce items with pre-specified variability that are hoped to have predictable levels of difficulty. It is somewhat unlikely that researchers utilizing these previous approaches would look at the proposed approach with favor; however, recent applications of machine learning show success in solving tasks that seemed impossible for machines not too long ago. The proposed approach uses deep learning to implement probabilistic language models, not unlike what Google brain and Amazon Alexa use for language processing and generation.

Key words: deep learning, neural networks, automatic item generation, machine learning.

1. Introduction

Utilizing algorithms to generate items in educational and psychological testing is an active area of research for obvious reasons: Test items are predominantly written by humans, in most cases by content experts who represent a limited and potentially costly resource. Using algorithms instead has the appeal to provide an unlimited resource for this crucial part of assessment development.

The approach presented here deviates in several ways from previous attempts to address the issue, while it is similar in that existing items are used as the basis for new items. In the past, automatic item generation (AIG) relied either on generating clones of narrowly defined item types such as those found in language free intelligence tests (e.g., Raven's progressive matrices) or used an analysis of cognitive task components of certain items, and derived schemata to produce new item instances (e.g., Bejar et al., 2003; Embretson, 2002; Embretson & Yang, 2007; Gierl & Lai, 2013). The solution presented here takes on this task by deep learning (DL), a neural network (NN)-based machine learning approach. It can be conjectured that researchers utilizing previous approaches would not look at this approach with favor; however, past success of DL in automated essay scoring and language generation show that the use of deep NNs (DNNs) is potentially able to move the field of AIG forward.

In the approach presented here, state-of-the-art recurrent neural network (RNN)-based language models (e.g., Mikolov, 2012) are used, similar to those now common in applications such as *Google brain* (https://research.google.com/teams/brain/) and Amazon *Alexa API* (https://developer.amazon.com/alexa). It is shown that this technique is able to generate items automatically.

Correspondence should be made to Matthias von Davier, National Board of Medical Examiners, 3750 Market Street, Philadelphia, PA 19104-3102, USA. Email: mvondavier@nbme.org

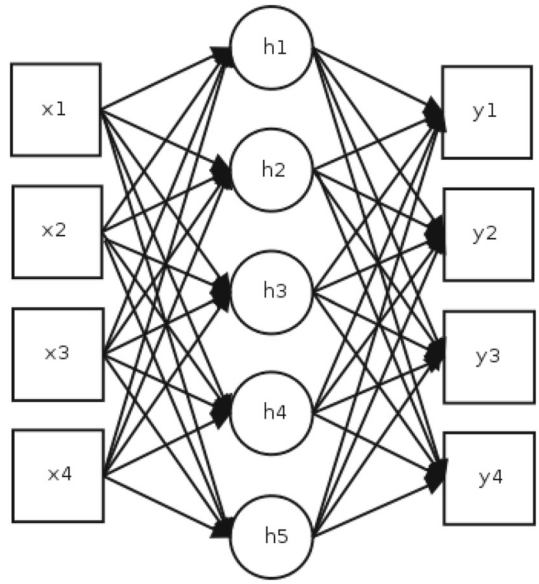

FIGURE 1.
Small feed-forward neural network with a single hidden layer.

Early artificial NN research is associated with the perceptron (Rosenblatt, 1958), which was developed in psychological science to describe a model for information storage and organization in the brain. Following this development, the field of artificial NN or neural computing was established and has grown into a major domain of machine learning used in fields such as computer vision, robotics, and natural language processing. Multilayer NNs and RNNs are often labeled as DL approaches, and the past few years have seen increasing use of these for language modeling, picture classification, face recognition, automated picture captioning, and other applications.

2. Learning in Neural Networks

NNs can be characterized as L layers of multiple real-valued variables (cells) h_{li}, typically arranged as vectors $\mathbf{h}_l = \left(h_{l1}, \ldots, h_{lk(l)}\right)$ for convenience of calculations and notation. Layer $\mathbf{h}_l$ receives input from the previous layer $\mathbf{h}_{l-1}$ and passes output to the next layer $\mathbf{h}_{l+1}$. The passing of information from an input layer, identified with $\mathbf{x} = \mathbf{h}_0$ to hidden layers $\mathbf{h}_l$ to output $\mathbf{y} = \mathbf{h}_L$, is carried out by what one may call a generalized linear model. NNs combine information in an additive (compensatory) manner and it can be shown that NNs can approximate arbitrary (including conjunctive and non-compensatory) functions (Cybenko, 1989; Hornik, 1991). NNs of this type are considered feed-forward networks, as there is a clear direction of information flow from input to output, just as there is a clear distinction between independent (input) variables and dependent (output). Figure 1 shows a small example of a NN with four cells in the input layer $\mathbf{x} = \mathbf{h_0}$, one hidden layer $\mathbf{h_1}$ with five cells and an output layer $\mathbf{y} = \mathbf{h_2}$ with four cells.

In particular, the input of layer $\mathbf{h}_{l-1} = \left(h_{l-1,1}, \ldots, h_{l-1,K(l-1)}\right)$ to the following cell h_{li} in layer $\mathbf{h}_l$ is calculated as

$$h_{li} = g_l\left[\sum_{j=1}^{k_{(l-1)}} w_{ji}^{[(l-1)l]} h_{(l-1)j} + b_{li}\right]$$

with some differentiable function $g_l : \mathbb{R} \mapsto]c_1, c_2[$ that maps the unbounded real value input to a compact open interval. Typical examples for g_l include the logistic function (or: sigmoid function in the language of NN research)

$$g_l(x) = \frac{1}{1 + \exp(-x)},$$

mapping real-valued input to the interval]0, 1[. Another function used in the design of NNs is the hyperbolic tangent function

$$g_l(x) = \tanh(x)$$

mapping real-valued input to the interval $]-1, +1[$. To give an example of how input $\boldsymbol{x}$ and output $\boldsymbol{y}$ is associated by learning in NNs, we will consider a simple network with only one hidden layer. In standard statistical language, we would denote $\boldsymbol{x}$ as independent and $\boldsymbol{y}$ as dependent variable. The free parameters of the network are the weights $w_{ij}^{[l-1,l]}$, which can be understood as regression parameters, and the biases $b_i^{[l]}$ for $l > 0$, which can be understood as intercepts.

For training, the NN is presented with a stream of input and output pairs $(\mathbf{x}_1, \mathbf{y}_1)$, $\ldots$, $(\mathbf{x}_t, \mathbf{y}_t)$, ..., $(\mathbf{x}_N, \mathbf{y}_N)$ and is asked to produce a predicted output vector $\hat{\mathbf{y}}_t(\mathbf{x}_t)$ for each input $\mathbf{x}_t$. Training is implemented as a minimization problem, for example by means of a quadratic loss function for which we wish to find

$$\arg\min_{\vec{\mathbf{w}}, \vec{\mathbf{b}}} \left\{ \sum_t \left(\mathbf{y}_t - \hat{\mathbf{y}}_t \left(\mathbf{x}_t; \vec{\mathbf{w}}, \vec{\mathbf{b}} \right) \right)^2 \right\}.$$

This defines an objective function to be minimized with respect to the network's parameters, the weights and biases, $\left(\vec{\mathbf{w}}, \vec{\mathbf{b}}\right)$. How this is done will be described briefly below in Sect. 2.2 on back-propagation.

2.1. Neural Networks for Discrete Data

For discrete data, including nominal data, another type of loss function is used for the minimization. Examples of nominal data are vocabularies or character sets (in natural language processing), or sets of permissible actions (in process data) or moves (in board games), or notes (in automatic music processing or music generation). The task to predict nominal data is a classification problem implemented using a coding scheme. What traditional statistics may call a dummy coding, machine learning researchers call 'one-hot' coding. In particular, a nominal output variable with V possible states $y \in \{o_1, \ldots, o_V\}$ is recoded such that $\boldsymbol{y}^* = (y_1, \ldots, y_V) \in \{0, 1\}^V$ with $y_i^* = 1$ if $y = o_i$ and $y_j^* = 0$ otherwise.

The output layer $\widehat{\boldsymbol{y}}^* = \boldsymbol{y}^*(\boldsymbol{x}) = \boldsymbol{h}_L$ represents the classification based on input data $\boldsymbol{x}$, which may be nominal (and dummy coded) as well, or can consist of data at any other scale level. The output layer produces the predicted $\widehat{\boldsymbol{y}}^*$ using a multinomial logistic function of the previous layer's input. This is to ensure that the sum of all predicted output variables equals 1. That is, we have

$$P\left(y_i^* = 1 \mid \mathbf{x}\right) = \frac{\exp\left(\sum_{j=1}^{k_{(l-1)}} w_{ji}^{[(l-1)L]} h_{(l-1)j} + b_{Li}\right)}{\sum_{m=1}^{V} \exp\left(\sum_{j=1}^{k_{(l-1)}} w_{jm}^{[(l-1)L]} h_{(l-1)j} + b_{Lm}\right)}.$$

The $\widehat{y}_i^* = P\left(y_i^* = 1 \mid \mathbf{x}\right)$ are the probabilities of the classification represented by the output nodes given input data $\mathbf{x}$. That is, the output can be interpreted as a probability distribution over a discrete (dummy coded) vocabulary. The following function of the observed values in the training sample, or better, in a cross-validation sample, and the predicted output defines a measure of discrepancy

$$-\sum_{i=1}^{V} y_i^* \log P\left(y_i^* = 1 \mid \mathbf{x}\right).$$

When looking at a series of inputs and outputs, for example a sample of N training or cross validation cases $(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)$, one can define

$$\text{loss} = \frac{1}{N}\sum_{n=1}^{N}\left[-\sum_{i=1}^{V} y_{in}^* \log P\left(y_i^* = 1 \mid \mathbf{x_n}\right)\right]$$

as a loss function, which is referred to as cross-entropy as this function relates to the measure of entropy $H(\mathbf{p}, \mathbf{q}) = -\sum_i p_i \log q_i$, as well as to what Gilula & Haberman (1994, 1995) call the log penalty (see also Savage, 1971), which is used to assess the quality of prediction functions.

2.2. *Back-Propagation*

NNs are typically trained using an algorithm that is known as back-propagation, originating in the 1960s and adapted for the training of neural networks (e.g., Rumelhart et al. 1986; Dreyfus, 1990). The basis of this optimization method is the principle of gradient descent, propagated using the chain rule of differentiation through the layers of the NN. Gradient descent methods, being iterative numerical optimization methods (e.g., Dennis & Schnabel, 1996), adjust the current parameter estimates by the gradient multiplied by (a typically small) constant, called the learning rate in this context.

The weights and biases are to be updated so that a suitable loss function, such as the ones described above, is minimized. In the case of a single hidden layer, we can write the hidden state as $\mathbf{h_1} = g\left(\mathbf{x}; \mathbf{w}^{x1}, \mathbf{b}^1\right)$ which describes $\mathbf{h_1}$ as a function of input layer $\mathbf{x}$. The output layer is $\mathbf{y}^* = f\left(\mathbf{h_1}; \mathbf{w}^{1y}, \mathbf{b}^y\right)$, the predicted value of the output as a function of the hidden layer. Taken together, we have a chained equation that describes the predicted output as a function of the input,

$$\mathbf{y}^* = f\left(\mathrm{g}\left(\mathbf{x}; \mathbf{w^{x1}}, \mathbf{b^1}\right); \mathbf{w}^{1y}, \mathbf{b}^y\right).$$

Therefore, the chain rule is needed when calculating the gradient of the parameter vectors of function g.

While a small network with a single hidden layer does not seem like a complex estimation problem, the largest NNs with multiple layers and many cells per layer approach billions of connections, coming close to biological counterparts (Trask et al. 2015), and hence with numbers of adjustable weight and bias parameters that far surpass classical statistical applications. Training NNs using back-propagation can be accelerated dramatically using modern parallel programming paradigms. Massively parallel algorithms have been developed for a variety of applications including computer vision (Cui, Wei & Dai, 2010), generalized latent variable models in psychometrics (von Davier, 2016, 2017) as well as machine learning applications utilizing NNs (Abadi et al., 2015).

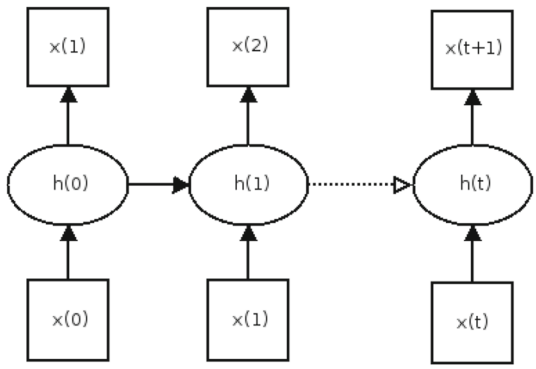

FIGURE 2.
A visualization of an unrolled recurrent neural network showing the hidden state dependency on previous hidden states.

3. RNNs, Deep Learning, and Language Models

RNNs are a type of NNs that turned out to be particularly suitable for modeling sequence data such as language, financial data, music, or process data. Instead of looking at the input stream alone, RNNs also take the current state of the network into account and integrate this with the current input. RNNs feed the input $\boldsymbol{x}_t$ from state t together with the hidden state $\mathbf{h}_t$ into the layer as concatenated input for the next state $t+1$. The sequence of observed states serves at the same time as observed input and desired next output state. More formally, we have a dependency on past observations

$$\mathbf{x}_t = f\left(\mathbf{x}_{t-1}, \mathbf{h}_{t-1}\right) + \mathbf{e}_t$$

with

$$\mathbf{h}_{t-1} = g\left(\mathbf{x}_{t-2}, \mathbf{x}_{t-3}, \ldots, \mathbf{x}_0\right).$$

The above equations express the current state $\mathbf{x}_t$ as a prediction function of the previous observed state $\mathbf{x}_{t-1}$ and the previous hidden layer state $\boldsymbol{h}_{t-1}$ plus an error term $\mathbf{e}_t$. The state or (internal) value of the hidden layer in turn can be viewed as a function of all previous observed values $\boldsymbol{x}_{t-2}, \ldots, \boldsymbol{x}_0$. This essentially produces a distribution of the next observation given all previous observations. Hence, the probability of a sequence can be calculated as

$$p\left(\boldsymbol{x}_0, \ldots, \boldsymbol{x}_t\right) = p\left(\boldsymbol{x}_0\right) \prod_{i=1}^{t} p\left(\boldsymbol{x}_i \mid \boldsymbol{x}_{i-1}, \ldots, \boldsymbol{x}_0\right).$$

While feed-forward NNs are general tools to fit arbitrary functions, RNNs seem ideally suited to fit arbitrary sequences (Schäfer & Zimmermann, 2006). In other words, the equation above can be understood as the network output in the form of a predicted probability of sequence $\boldsymbol{x}_0, \ldots, \boldsymbol{x}_t$. Training the network essentially means minimizing prediction error by maximizing the probability of the training sequences.

Figure 2 presents a visualization of a recurrent neural network with one hidden layer. Note how the current cell state h_t depends on the previous state h_{t-1} and the current input x_t.

Training RNNs proved to be challenging as the chain rule used in back-propagation applied repeatedly in the dependencies of current states on previous states led to phenomena known as *exploding* or *vanishing* gradients. This made training of networks with longer chains of dependencies between states practically impossible. The introduction of a more complex network cell structure, proposed by Hochreiter & Schmidthuber (1997), called the long short-term memory (LSTM) cell solved this problem. The LSTM cell contains a hidden state that is being fed back in a cyclic graph, but this feedback is moderated by a forget gate, and the passing through of previous state and internal state is additionally moderated by an output gate. LSTM cells themselves are implemented using 4 layers, but the combination of results is not just additive and different inputs and outputs may be combined in a multiplicative way (after being transformed by logistic or hyperbolic tangent function).

It has been shown that LSTM networks with suitable training algorithms are able to solve the vanishing/exploding gradient issue. Moreover, LSTM and variants proposed to solve the same issue in different ways are now part of the NN programming toolkits freely provided by Google and others and have been used for language modeling (Mikolov, 2012; Sundermeyer et al., 2015; Jozefowicz et al. 2016). More details about LSTM networks and a systematic comparisons of variants of LSTM and other approaches to solving the exploding/vanishing gradient problem in networks with sequential memory can be found in Greff et al. (2015) and Jozefowicz et al. (2015).

LSTM-based RNNs are currently considered a staple of artificial intelligence-based language modeling (Jozefowicz et al. 2016; Sundermeyer et al. 2015), and their performance appears to be somewhat surprising or might even seem unreasonable to some (Karpathy, 2015), given that no explicit model of the language structure is implemented but rather inferred from a corpus of texts provided as training data.

4. Automated Item Generation Using Deep Learning

Language models based on RNNs are commonly used for language generation, automated translations, and captioning. Therefore, applying this tool to generate items based on a corpus of existing items can be considered a promising endeavor. There are potential limitations, however, as items in a questionnaire or a cognitive test may be not suitable for generating new items without further review and selection. Once accomplished, the ability to perpetually churn out new item prototypes with a trained RNN makes this less of an issue, as one can argue that text can be easily discarded if it does not pass the muster. While automatic item generation using language models, like any other approach, can likely not be conducted without human review of the generated items, this tool can provide a continuous supply of item candidates.

4.1. Models and Tools

The models initially explored in this study were different multilayer LSTM-RNN, ranging from 256 cells per layer and 2 hidden layers to 64 cells per layer and 4 hidden layers. This paper will not provide a systematic comparison of these network architectures, as a comparison would require direct measures of performance relative to the sequences that were generated. All networks tried in this exercise were able to be trained to about the same level of predictive power as expressed by the cross-entropy reached by training with the data described below. The calculations were done in Python, using the TensorFlow library, the open DL toolkit provided by the Google Brain team (Abadi et al., 2015), and the LSTM-RNN was specified based on the char-rnn scripts (Ozair, 2016) which are a Python implementation of a toolkit originally developed for another DL library, Torch (Karpathy, 2015). The default setting of the char-rnn scripts was used, the scripts are available at github (Ozair, 2016), and the source material for the training can be obtained from the author upon request.

Training took place on a Dell T7500 workstation equipped with a Nvidia GeForce GTX 1070 GPU; the GPU was accessed using the CUDA 8.0 and cuDNN 5.1 toolkits through TensorFlow. The training was done using the Adam Optimizer (Kingma & Ba, 2014), a stochastic steepest descent method.

4.2. Item Corpus

The items used in the current analyses are based on feeding personality items that are publicly available online to an RNN and training the network as described above. One major source of public domain personality items is the International Personality Item Pool (IPIP; Goldberg, 1999; Goldberg et al. 2006) pool containing 3,320 items available at http://ipip.ori.org/AlphabeticalItemList.htm. Since RNNs need large amounts of data for training probabilistic sequential dependencies, this pool is a good starting point for experimentation with AIG using RNNs. The example items that were selected for this study were generated automatically by a DNN using a 64-cell recurrent network with 4 hidden layers. The training sequence length was set at 256 characters, and other parameters such as the ones influencing dropouts (Gal & Ghahramani, 2015) were not modified. The size of the vocabulary was 68 unique characters. Training was terminated once the cross-entropy (or log penalty; Gilula & Haberman, 1994) reached a value of slightly below 0.23, which corresponds to an average probability of subsequent characters given the previous sequence of about $p = 0.8$ (since $-ln\,(0.8) \sim 0.23$). Given the relatively small size of the corpus, this ensured that most generated items consist of correctly spelled words, while a network trained using a small corpus at this level of accuracy does not produce too many exact clones of human generated items contained in the corpus.

4.3. Results and Performance of Generated Items

Automatically generated items underwent an empirical study to examine whether they differ from existing items in terms of psychometric qualities.[1] A selection of 24 automatically generated items was combined with a total of 17 items from publicly available sources and adapted from published short versions of 5-factor inventories (IPIP; Goldberg, 1999; Goldberg et al. 2006; Rammstedt & John, 2007). An online test was assembled that comprised of these 24+17= 41 items for a data collection using Google Forms for online delivery and Amazon Turk for recruiting test takers. Items were presented in random order for each participant, without any indication whether these were generated automatically, or taken from existing instruments, or whether they are expected to belong to one or another scale.

Amazon Turk allows delivery of assignments to anonymous workers across the globe. Each worker obtains a link to the questionnaire and, upon completion, receives a secret code to enter into the Amazon Turk site in order to obtain payment. An amount of US$0.25 was offered, and no requirement other than understanding the English language to operate as Amazon Turk worker was imposed.

Test takers were instructed to fill out a "new form of personality test" and were informed that the expected time to finish the assignment was 7 min. However, it took respondents on average only 4 min and 30 s on average to complete the questionnaire. The online questionnaire contained only the 41 items and did not ask for any other information given the limitations of what can be asked for if given just a nominal payment. In total, $N = 277$ respondents completed the questionnaire through Amazon Turk.

[1] Thanks to the excellent suggestion received from reviewers of the first draft, it was decided to collect actual response data using automatically generated items and compare these to response data from published human generated personality items. Additional experiments with dropouts, another suggestion received from reviewers, which allow to train networks with a form of regularization, will be conducted in future research.

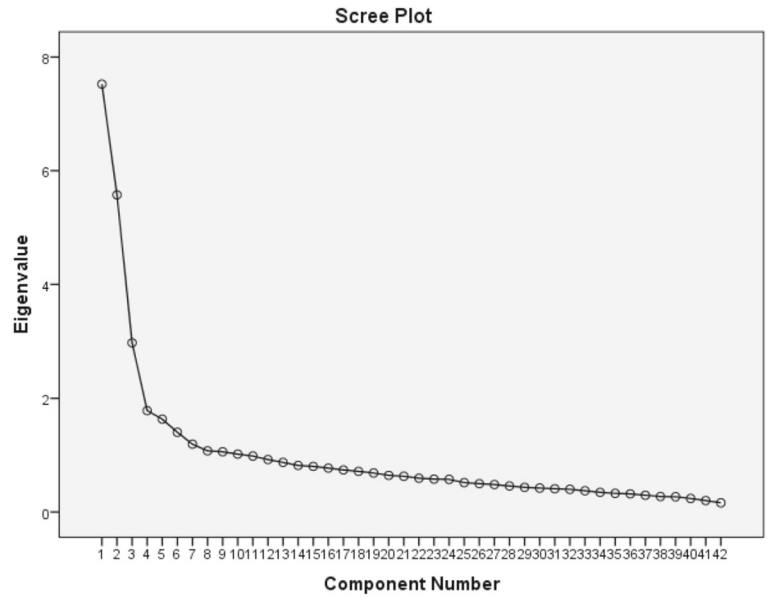

FIGURE 3.
Scree plot indicating 4 or more factors. The solution chosen is based on extraction of 5 factors as this is a number that can be expected based on the origin of the source item material.

The raw data were checked for omissions and otherwise incomplete response patterns. However, apart from two missing responses, all test takers provided complete data. There was no indication that automatically generated items were substantially harder or easier to endorse than human authored items.

In order to examine whether the generated items behave similarly compared to human authored items in terms of internal structure, the data were analyzed using SPSS 23's factor analysis module. The method chosen was an exploratory factor analysis with the number of extracted factors set to five, as the source items came from 5-factor inventories. Figure 3 shows the scree plot for the data.

After extraction, the solution was Oblimin rotated with default settings provided by SPSS, and results were printed with ordered loadings and while suppressing all loadings smaller than 0.3.

The resulting pattern matrix is provided in Table 1, which also contains the item stem and a column that indicates whether the item was automatically generated (AIG). It is apparent that the location of highest factor loadings is based on item content, and similarities among items, rather than on whether items were generated automatically or not.

This result is indication that the automatically generated and the original items can be grouped according to factors or constructs they tap into in very similar ways. Moreover, the resulting factors are largely in agreement with the 5-factor solution that is commonly found in these personality inventories. Factor 1 can be associated with neuroticism, factor 2 with extraversion, factor 3 with conscientiousness, 4 with agreeableness, and finally factor 5 with openness. The number of item loadings > 0.3 for 40 of the 41 items is either unique to one factor or may occasionally show a secondary loading. The appearance of secondary loadings larger than 0.3 does not seem to be associated with the AIG variable. Only item 20 ("I think I would not be a success in movies"), which was automatically generated, does not appear to have a clear association with a factor.

TABLE 1.
Pattern matrix for the 5-factor solution based on real data.

Factor					Item stem	AIG
1	2	3	4	5		
0.66					I often do not know what happens to me	1
0.66					I rarely consider my actions	1
0.60				− 0.35	I worry so much about myself	1
0.59					I am usually skeptical about myself	1
0.58					I am unable to deal with strange experiences	1
0.57					I believe that most people like always betrayed me	1
0.56					I don't know what I really lack	1
0.56					I don't think about decisions	1
0.54				− 0.45	I get nervous easily	
0.51					I check on things out of nothing	1
0.51					I plan my life based on other people	1
0.46					I know that I am troubled by praise	1
0.43	0.41				I usually control others	1
0.36					I work hard to get others to do things	1
0.35					I tend to be lazy	
0.31					I have few artistic interests	
0.31					I go out of my way to meet outside advice	1
	− 0.58				I don't like to draw attention to myself	
	− 0.57				I am a reserved person	
	0.57				I talk to a lot of different people at parties	
	0.57		0.31		I am outgoing and sociable	
	0.45				I prefer to be the perfect leader	1
− 0.09	− 0.23	− 0.01	− 0.08	− 0.03	I think I would not be a success in movies	1
		0.67			I am willing to take responsibility for my work	1
		0.60			I pay attention to details	
		0.57			I tend to do a thorough job	
		0.54			I enjoy thinking about things	
− 0.34		0.52			I am aware of how I am doing	1
		0.49	0.32		I accept people as they are	
		0.43			I am about my motives	1
		0.43			I have an active imagination	
		0.42			I make decisions in my life where other people seem to avoid it	1
0.36			− 0.55		I tend to find fault with others	
			0.52		I am a trusting person	
0.36			− 0.40		I feel that I am too blunt	1
			− 0.34		I am not interested in other people's problems	
				0.76	I worry so little	1
				0.57	I am seldom bothered by problems	1
				0.48	I can handle stress well	
				0.40	I am not easily annoyed	
	0.33			0.35	I can make any setting my situation	1

Loadings smaller than 0.3 (absolute) are suppressed and items are ordered based on the absolute size of the loadings. This solution is based on 24 automatically generated items and 17 items that are sourced from publicly available item banks.

It appears that the automatically generated items selected for this data collection do not differ systematically from items selected from a human generated item pool. This implies that RNN-based automatically generated items could be a valuable source for replenishing item pools for personality domains. Preselection of items could be conducted with the data collections used here. Even though the incentive for completion was quite modest, the data quality, judging from the results presented here, appears to be quite satisfactory.

5. Discussion

The present paper explores automated item generation using DNNs trained using a publicly available database of personality test items. With more item examples, and more complex network structure, the ability to generate plausible item candidates for human editing can be expected to increase.

Several of the example items that were generated have the appeal of, or may even pass the muster of an item one may expect to see in a Five-Factor inventory of personality. However, the selected examples, while impressive (at least for some), are just that, a selection, and further work is needed to refine the methodology, to fine tune training and potentially to include more variables into the language model implemented as a multilayer RNN. One fruitful direction might be a recent development that allows the use of variational auto-encoders (VAE) that allow to enhance recurrent networks by adding latent variables representing further context of an item (Chung et al. 2015), another may be the use of generative adversarial networks (GANs; Goodfellow et al., 2014), a method that may be potentially useful to evaluate network output combined with semi-unsupervised learning of the quality of generated items to further improve training the network used for AIG. Additional work is needed to extend this first exploration to more complex item stems and potentially the automatic generation of distractors for a given item stem.

The use of more complex NN models requires more research on coding of context variables and on training increasingly complex network structures. However, these initial results are rather encouraging as the selection of NN-based automatically generated items appears not to differ substantially from items found in publicly available sources.

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jozefowicz, R., Jia, Y., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Schuster, M., Monga, R., Moore, S., Murray, D., Olah, C., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., & Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org (Google Research).

Bejar, I. I., Lawless, R., Morley, M. E., Wagner, M. E., Bennett, R. E., & Revuelta, J. (2003). A feasibility study of on-the-fly item generation in adaptive testing. *Journal of Technology, Learning, and Assessment*. https://www.uam.es/personal_pdi/psicologia/fjabad/cv/articulos/jlta/A_Feasibility_Study_of_On_the_Fly_Item_Generation_in_Adaptive_Tes%5B1%5D.pdf. Accessed 7 March 2018.

Chung, J., Kastner, K., Dinh, L., Goel, K., Courville, A., & Bengio, Y. (2015). *A recurrent latent variable model for sequential data*. arXiv:1506.02216v6 [cs.LG].

Cui, H., Wei, X., & Dai, M. (2010). Parallel implementation of expectation-maximization for fast convergence. In *ACM proceedings*. http://users.ece.cmu.edu/~hengganc/archive/report/final.pdf. Accessed 7 March 2018.

Cybenko, G. (1989). Approximations by superpositions of sigmoidal functions. *Mathematics of Control, Signals, and Systems*, *2*(4), 303–314.

Dennis, J. E., & Schnabel, R. B. (1996). Numerical methods for unconstrained optimization and nonlinear equations. *Classics in Applied Mathematics: Society for Industrial and Applied Mathematics*. https://doi.org/10.1137/1.9781611971200.

Dreyfus, S. E. (1990). Artificial neural networks, back propagation, and the Kelley–Bryson gradient procedure. *Journal of Guidance, Control, and Dynamics*, *13*(5), 926–928.

Embretson, S. E. (2002). Generating abstract reasoning items with cognitive theory. In S. H. Irvine & P. C. Kyllonen (Eds.), *Item generation for test development* (p. 219250). Mahwah, NJ: Erlbaum.

Embretson, S. E., & Yang, X. (2007). Automatic item generation and cognitive psychology. In C. R. Rao & S. Sinharay (Eds.), *Handbook of Statistics: Psychometrics* (Vol. 26, p. 747768). North Holland: Elsevier.

Gal, Y., & Ghahramani, Z. (2015). A theoretically grounded application of dropout in recurrent neural networks. Published in NIPS 2016. arXiv:1512.05287

Gierl, M. J., & Lai, H. (2013). Using automated processes to generate test items. *Educational Measurement: Issues and Practice*, *32*, 3650.

Gilula, Z., & Haberman, S. J. (1994). Models for analyzing categorical panel data. *Journal of the American Statistical Association*, *89*, 645–656.

Gilula, Z., & Haberman, S. J. (1995). Prediction functions for categorical panel data. *The Annals of Statistics*, *23*, 1130–1142.

Goldberg, L. R. (1999). A broad-bandwidth, public domain, personality inventory measuring the lower-level facets of several five-factor models. In I. Mervielde, I. Deary, F. De Fruyt, & F. Ostendorf (Eds.), *Personality psychology in Europe* (Vol. 7, pp. 7–28). Tilburg: Tilburg University Press.

Goldberg, L. R., Johnson, J. A., Eber, H. W., Hogan, R., Ashton, M. C., Cloninger, C. R., et al. (2006). The international personality item pool and the future of public-domain personality measures. *Journal of Research in Personality*, *40*, 84–96.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, J. (2014). *Generative adversarial networks*. arXiv:1406.2661.

Greff, K., Srivastava, R. K., Koutnik, J., Steunebrink, B. R., & Schmidhuber, J. (2015). *LSTM: A search space odyssey*. arXiv preprint arXiv:1503.04069.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*(8), 17351780.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural Networks*, *4*(2), 251–257. https://doi.org/10.1016/0893-6080(91)90009-T.

Jozefowicz, R., Vinyals, O., Schuster, M., Shazeer N., & Wu, Y. (2016). *Exploring the limits of language modeling*. arXiv:1602.02410v2.

Jozefowicz, R., Zaremba, W., & Sutskever, I. (2015). An empirical exploration of recurrent network architectures. In *Proceedings of the 32nd international conference on machine learning, Lille, France* (Vol. 37). JMLR: W&CP.

Karpathy, A. (2015). *The unreasonable effectiveness of RNNs*. http://karpathy.Github.io/2015/05/21/rnn-effectiveness/. Accessed 7 March 2018.

Kingma, D., & Ba, J. (2014). *Adam: A method for stochastic optimization*. arXiv preprint arXiv:1412.6980.

Mikolov, T. (2012). *Statistical language models based on NNs*. Ph.D. thesis, Brno University of Technology.

Ozair, S. (2016). *Char-rnn for tensorflow*. https://github.com/sherjilozair/char-rnn-tensorflow. Accessed 7 March 2018.

Rammstedt, B., & John, O. P. (2007). Measuring personality in one minute or less: A 10-item short version of the big five inventory in English and German. *Journal of Research in Personality*, *41*, 203–212. https://doi.org/10.1016/j.jrp.2006.02.001.

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

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). *Learning internal representations by error propagation* (Vol. 1). Cambridge, MA: MIT press.

Savage, L. (1971). Elicitation of personal probabilities and expectations. *Journal of the American Statistical Association*, *66*(336), 783–801. https://doi.org/10.2307/2284229.

Schäfer, A. M., & Zimmermann, H. G. (2006). Recurrent neural networks are universal approximators. In S. D. Kollias, A. Stafylopatis, W. Duch, & E. Oja (Eds.), *Artificial neural networks— ICANN 2006. ICANN 2006. Lecture notes in computer science* (Vol. 4131). Berlin: Springer.

Sundermeyer, M., Ney, H., & Schlüter, R. (2015). From feedforward to recurrent LSTM NNs for language modeling. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, *23*(3), 517–529. https://doi.org/10.1109/TASLP.2015.2400218.

Trask, A., Gilmore, D., & Russell, M. (2015). *Modeling order in neural word embeddings at scale*. CoRR, abs/1506.02338, 2015. arXiv:1506.02338.

von Davier, M. (2016). High-performance psychometrics: The parallel-E parallel-M algorithm for generalized latent variable models. *ETS Research Report Series*, *2016*, 111. https://doi.org/10.1002/ets2.12120.

von Davier, M. (2017). New results on an improved parallel EM algorithm for estimating generalized latent variable models. In L. A. van der Ark, M. Wiberg, S. A. Culpepper, J. A. Douglas, & W.-C. Wang (Eds.) *Quantitative psychology: Proceedings of the 81st annual meeting of the psychometric society, Asheville, North Carolina, 2016* (p. 1–8). http://www.springer.com/us/book/9783319562933.

Manuscript Received: 23 MAR 2017
Final Version Received: 27 DEC 2017
Published Online Date: 12 MAR 2018