Technical Report PAR-10021867-092020-2

Multilayer perceptron neural networks as multi-variable material models

Biswajit Banerjee
Parresia Research Limited, Auckland, New Zealand
b.banerjee.nz@gmail.com

David M. Fox
CCDC Army Research Laboratory, Aberdeen Proving Ground, MD, USA
david.m.fox1.civ@mail.mil

Richard A. Regueiro
University of Colorado, Boulder, CO, USA
richard.regueiro@colorado.edu

Tuesday, 29 September, 2020

Abstract

Material models that depend on multiple independent variables are often necessary for accurate numerical simulations, particularly for applications that involve large stresses and deformations. It is rare that purely physics-based models are used in simulations because of the attendant computational cost. Instead, experimental and microscale simulation data are expressed as phenomenological models and fed into simulations. As the number of independent variables increases, such models are not only difficult to design but also need exponentially larger amounts of data to parameterize accurately. In this paper we examine an alternative procedure for developing multi-variable phenomenological models via multi-layer perceptron neural networks. The method is applied to experimental data crush-curve and bulk modulus data for dry concrete sand. We find that reasonable network topologies and selected optimization parameters can be discovered by a factorial design of experiments procedure. Networks containing rectified linear unit (ReLU) activation functions are observed to produce excellent fits to the input data as well as acceptable generalization provided derivatives of the neural network models are not required in simulations.

1 Introduction

Experimental stress-strain data for granular materials often exhibit elastic-plastic coupling where the elastic moduli depend strongly on the amount of plastic strain. The moduli also depend on the initial density of the material. For high strain-rate processes in fully or partially saturated granular media, the moduli additionally vary with the porosity and degree of saturation. Similar dependence of material properties on multiple, apparently independent, variables are observed for yield limit surfaces and also the direction of the plastic strain rate (Brannon et al., 2015; Banerjee and Brannon, 2017; Banerjee and Brannon, 2019).

Numerical simulations of such materials are typically performed with constitutive models that contain a mix of phenomenological and physics-based models for the parts. In this paper we are concerned with phenomenological models that depend on multiple variables. In particular, we focus on the model of a material that unloads elastically in volumetric compression and can be represented in that regime by a mean stress (p) model of the form $p = p(\varepsilon_v^e, \varepsilon_v^p)$ where ε_v^e is the elastic volumetric strain and ε_v^p is the plastic volumetric strain. The bulk modulus (K) can be extracted from the model for p and has the form $K = K(\varepsilon_v^e, \varepsilon_v^p)$. An alternative form that is commonly used is $K = K(I_1, \varepsilon_p)$ where $I_1 = 3p$.

The design of closed-form functional expressions for p or K can be non-trivial and time consuming. A different functional form is required for each material that is tested. For instance, two nominally identical natural sands can exhibit remarkably different coupled elastic-plastic behavior. Also, the process of determining the parameters of these models from multidimensional experimental data is involved even for only two independent variables. Therefore, a unified and consistent method of designing material models is required.

In recent years, developments in machine learning have introduced encouraging possibilities for model design. These methods eliminate the model design process to some extent and focus instead on parameter determination. We have explored the application of support vector machines to data for a poorly-sorted concrete sand in Banerjee, Fox, and Regueiro (2020) and found that the models are reasonably accurate but could be improved. In this paper, we study multilayer perceptron (MLP) artificial neural networks and apply it to the same sand data. We find that MLP models can be used to fit the input data extremely well; in fact, better than support vector machines. Rectified linear unit (ReLU) activation functions are found to perform well, particularly when derivatives of the model function are not required in simulations. Direct fits to bulk modulus data are found to generalize better than bulk moduli computed from fits to pressure data. We also explore the effect of MLP training parameter variation on the fitting process to crush-curve data to obtain a better appreciation of the adequacy of MLP neural networks.

This paper is organized as follows. In Section 2 we briefly describe the theory behind multilayer perceptron networks. A summary of the experimental data is provided in Section 3. MLP fits to crush-curve data are discussed in Section 4 while those to pressure and bulk modulus data are examined in Section 5. Some concluding remarks are presented in Section 6.

2 Multilayer perceptron neural networks

Multilayer perceptron (MLP) neural networks (Rumelhart, Hinton, and Williams, 1985; Haykin, 1994) are a class of graph models that form the foundation for several machine learning models that have recently regained popularity. A schematic of a typical multilayer neural network is shown in Figure 1. For the topology shown in the schematic, the network is fed one sample at a time and the weights are adjusted to minimize the error in the prediction. In this particular example, the input consists of a vector $\mathbf{x}$, the connection weights $\boldsymbol{W}$, and the bias weights $\mathbf{b}$:

$$\mathbf{x} = \begin{bmatrix} x^1 \\ x^2 \end{bmatrix}, \quad \boldsymbol{W} = \begin{bmatrix} w^{11} & w^{12} \\ w^{21} & w^{22} \\ w^{31} & w^{32} \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b^1 \\ b^2 \\ b^3 \end{bmatrix}. \tag{1}$$

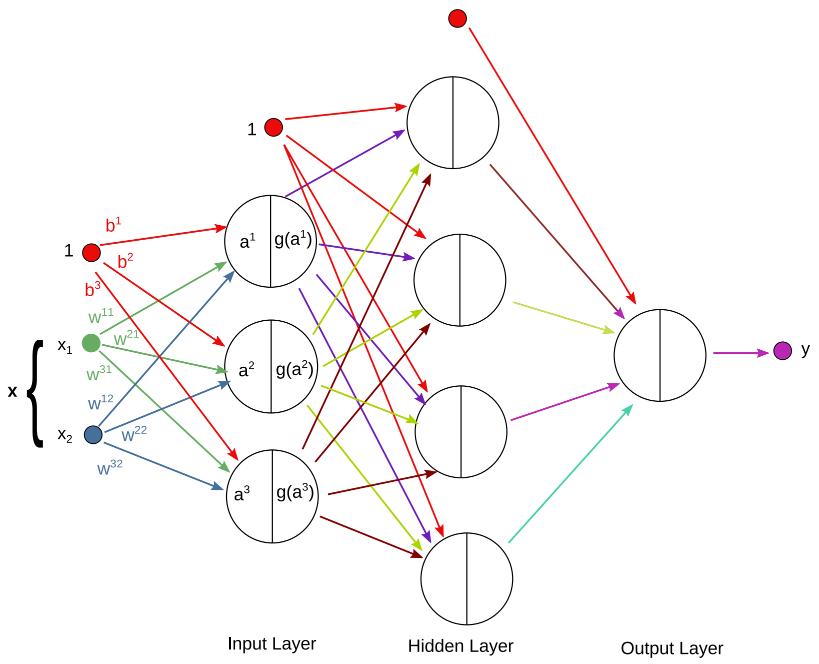


Figure 1 – *Schematic of a multilayer perceptron neural network.*

Nodes in the input layer are assigned weights

$$\mathbf{a} = \boldsymbol{W} \cdot \mathbf{x} + \mathbf{b} \quad \Longrightarrow \quad \begin{bmatrix} a^1 \\ a^2 \\ a^3 \end{bmatrix} = \begin{bmatrix} w^{11}x^1 + w^{12}x^2 + b^1 \\ w^{21}x^1 + w^{22}x^2 + b^2 \\ w^{31}x^1 + w^{32}x^2 + b^3 \end{bmatrix}. \tag{2}$$

An activation function $\mathbf{g}(\mathbf{a})$ determines the values that are passed on as inputs to the next layer, and the process is repeated until the output layer is reached.

In general, input samples are can be assumed to be vectors of the form $\mathbf{x} = (x^1, x^2, \ldots, x^d)$ and the output is a vector $\mathbf{y} = (y^1, y^2, \ldots, y^p)$, i.e., we seek a map from $\mathbb{R}^d$ to $\mathbb{R}^p$. The network is fully connected. Each pair of layers has an associated weight matrix $\boldsymbol{W}$ containing components w_i^j where j is the index of a node in the current layer and i is the index of a node in the previous layer. There is also a set of bias terms $\mathbf{b} = (b^1, b^2, \ldots, b^n)$ for each layer. The weights and biases are combined together in each layer and subjected to a nonlinear transformation $\mathbf{g}$, typically of sigmoidal or ReLU form (Hahnloser et al., 2000), and the output is passed on to the next layer. Because of the application of the activation function, a node of the graph is more commonly known as a neuron.

The functional form of this map for a single hidden layer network is

$$\mathbf{y} = f(\mathbf{x}) = \mathbf{g}_2(W_2 \cdot \mathbf{g}_1(W_1 \cdot \mathbf{g}_0(W_0 \cdot \mathbf{x} + \mathbf{b}_0) + \mathbf{b}_1) + \mathbf{b}_2) \tag{3}$$

where $\boldsymbol{W}_i$ are weight matrices, $\mathbf{b}_i$ are the bias vectors, and $\mathbf{g}_i$ are the activation operators for layer i. Note that the last operator in the sequence is typically scalar-valued for the models we consider in this paper. Multiple hidden layer networks can be composed using the same approach.

The weights are computed by feeding each sample into the network (typically in batches of 32 or 64), solving a least squares minimization problem using gradient descent with backpropagation, and refining the weights by randomly shuffling the training data over multiple epochs. Consider the situation where the input data are of the form $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_m, y_m)\} \subset \mathbb{R}^d \times \mathbb{R}$. The goal of a MLP model is to find a function $y = f(\mathbf{x})$ that fits the data such that the error is minimized:

$$\underset{\boldsymbol{W},\mathbf{b}}{\text{minimize}} \quad \tfrac{1}{2} \sum_{k=1}^{m_{\text{batch}}} \lVert f(\mathbf{x}_k) - y_k \rVert^2 \tag{4}$$

Because of the recursive nature of the MLP approximation in equation (3), gradients are computed by reverse-mode automatic differentiation (Rall and Corliss, 1996), popularly called backpropagation.

There are several tuneable parameters in the optimization process, most of which can be estimated automatically using modern software such as Keras (Gulli and Pal, 2017) and Tensorflow (Abadi et al., 2016). Accuracy depends strongly on the choice of the nonlinear transformation (also called an activation function).

3 Experimental data

The experimental data that have been used as a test-bed for the modeling process are for a dry, poorly-graded, concrete sand described by Fox et al. (2014) and tested at the University of Maryland.[1] Further details on the particular set used in this work can be found in (Banerjee, Fox, and Regueiro, 2020).

The hydrostatic loading-unloading data for that sand can be seen in Figure 2(a). The loading curve, shown in green, is used to fit a crush-curve model. The unloading curves are used to fit a bulk modulus model that depends on the plastic strain. Tangents to the unloading curves represent the bulk modulus and have been plotted in Figure 2(b).

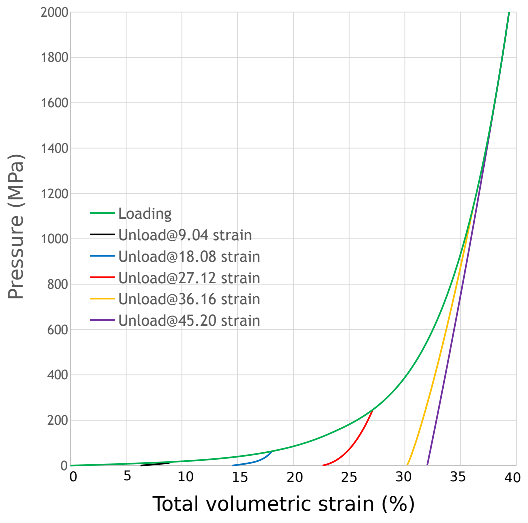
The crush-curve extracted from the hydrostatic compression data is depicted in Figure 3(a). Since the term "crush-curve" is used more commonly to refer to the change in porosity as a function of pressure form, we shown this form of the curve in Figure 3(b). The porosity (ϕ) has been computed using $\phi = p_3 - \varepsilon_p^v$ where ε_p^v is the volumetric plastic strain and $p_3 = 0.325$ is the volumetric plastic strain at which all pores have been crushed (Brannon et al., 2015).

4 Crush-curve model

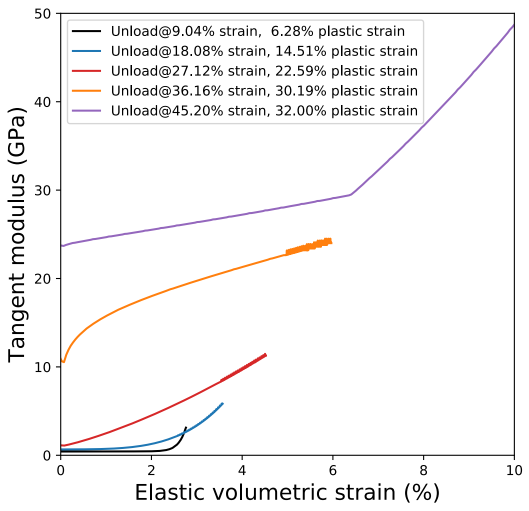
The crush-curve data available for the sand depends only on the volumetric plastic strain, and is therefore easier to model. Of course, crush-curves may depend on other variables such as the water content or some damage parameter, in which case the modeling process is more involved.[2]

[1]Stephen Akers, 2018, Private communication, CCDC Army Research Laboratory, Aberdeen Proving Ground, MD, USA

[2] If the crush curve depends only on a single independent variable, it is preferable to use a tabular model if we seek to approximate the experimental data better and to evaluate the model efficiently.

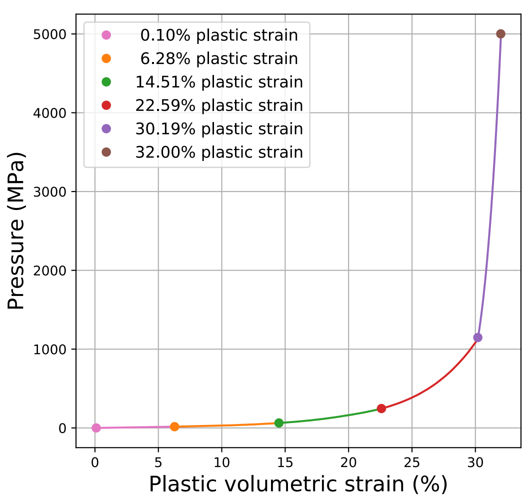


(a) Loading-unloading curves

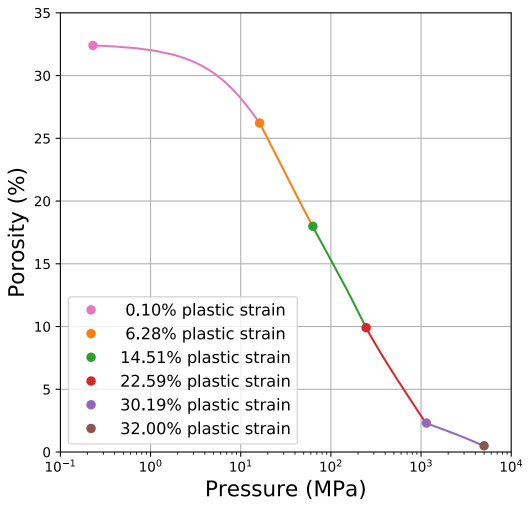


(b) Tangent bulk modulus curves

Figure 2 – *Hydrostatic loading-unloading data and unloading bulk moduli for dry poorly-graded concrete sand.*



(a) Pressure crush-curve.



(b) Porosity crush-curve.

Figure 3 – *Crush-curves for dry poorly-graded concrete sand.*

Before the data are modeled with a neural net, the input data need be scaled to that gradient computations do not lead to floating point precision errors. From Figures 3(a) and (b) it may appear that the scaling procedure can be identical for the two representations of the data. However, we have found that not to be the case. In particular, for the monotonically increasing curve, no fit is computed if the data are scaled linearly between 0 and 1. A reasonable fit is produced only if the data are centered around the mean and scaled to unit variance. Note that this is different from the behavior observed for support vector regression of the same data (Banerjee, Fox, and Regueiro, 2020).

The pandas 0.25.3 package (McKinney, 2011) was used to preprocess the data and scikit-learn 0.23.2 (Pedregosa et al., 2011) was used to scale the data. During the fitting process, arrays were managed using the NumPy 1.17.4 package (Harris et al., 2020) and the neural network model was implemented and solved with the Keras frontend (Gulli and Pal, 2017) for Tensorflow 2.0.0 (Abadi et al., 2016). All computations were performed in double precision.[3] For reproduceability of the results, a random number seed of 42 was used for NumPy, 1234 for Tensorflow, and 12345 for the Python random number generator. The number of input data points for the crush-curve data is 950.

If mechanics simulations, the crush-curve model is expected to be evaluated numerous times per particle (or per Gauss point) during each timestep. Also, for accuracy requires a large number of timesteps - particularly for high-rate processes. Therefore, an appropriate MLP model should have as low a computational cost as possible, i.e., the minimum number of layers and neurons per layer, and a low-cost activation function, needed to achieve a desired level of accuracy.

4.1 One-layer model

We choose a one-layer neural network consisting of 32 neurons in the hidden layer that are densely connected to a single input and produce a single output. The weights are initialized using a uniform random number generator. A ReLU activation function is used for the hidden layer:

$$\mathbf{g}(\mathbf{a}) := R(\boldsymbol{W} \cdot \mathbf{x} + \mathbf{b}) = \max(\mathbf{0}, \boldsymbol{W} \cdot \mathbf{x} + \mathbf{b})\,. \tag{5}$$

If we use this model, and allow Tensorflow to automatically choose the hyperparameters of the gradient descent process, we are required onlly to control the batch size and the number of training epochs. For a batch size of 128 and with 800 training epochs, we get the fits shown in Figure 4. Notice that the pressure vs. volumetric plastic strain curve appears to be fit better than the porosity vs, pressure crush-curve.

We can vary the number of samples per batch to observe the effect of batch size on the quality of the fit as is seen in Figure 5. The relative error in the figures is defined as

$$\text{Error} = 2\frac{y_{\text{pred}} - y_{\text{expt}}}{|y_{\text{pred}}| + |y_{\text{expt}}|} \tag{6}$$

The optimal batch size can be determined by examining the minimum, mean, and standard deviation of the error. For the pressure vs. plastic strain curve, the minimum absolute error is for a batch size of 128. This batch size also has the least mean absolute error is 24.6% with a standard deviation of 48.5%. On the other hand, though the minimum error for the porosity vs. pressure curve is also

[3]If care is not taken to use double precision for computation, the model may produce different results when imported into a mechanics simulation code.

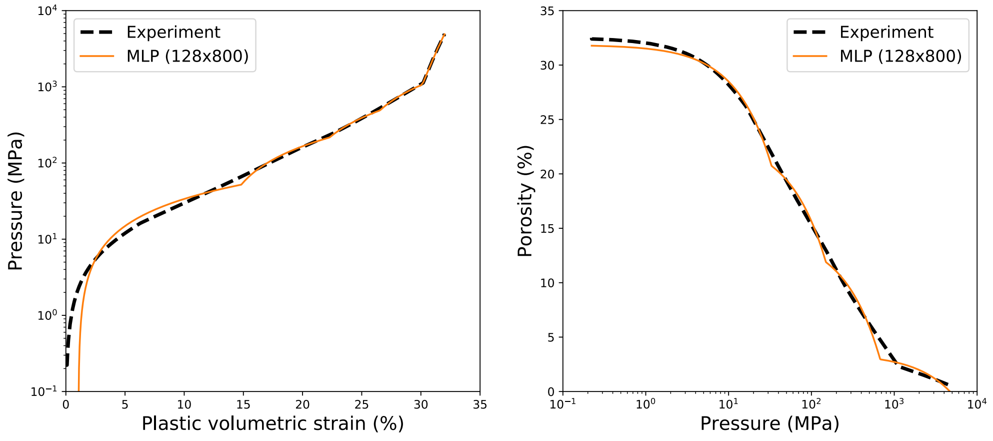


Figure 4 – *Single layer perceptron neural network fits to the crush-curve for poorly-graded concerete sand.*

attained for a batch size of 64, the mean absolute error is least for a batch size of 16 (4.9%) as is the standard deviation (7.7%).

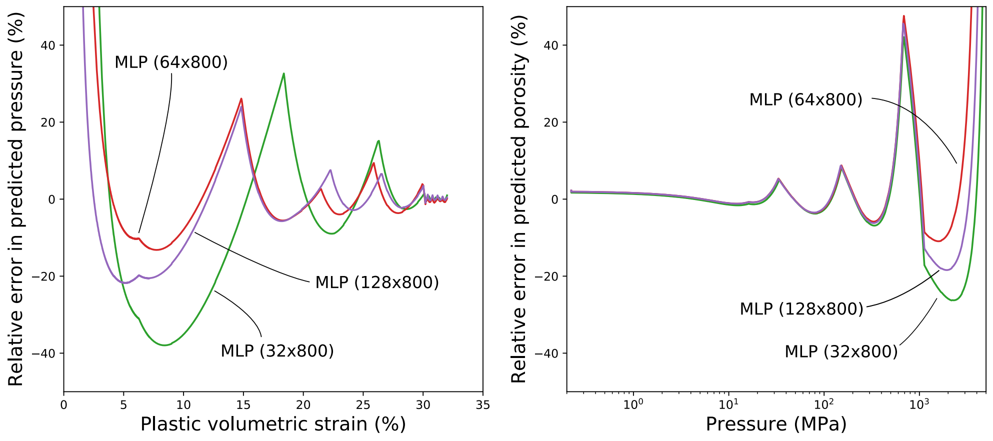


Figure 5 – *Relative prediction error for one-layer neural network fits to the crush-curve for varying batch sizes and 800 training epochs.*

If we vary the number of training epochs while keeping the batch size fixed at 128, we obtain the error curves in Figure 6. In this case, the minimum (as well as the smallest mean and standard deviation) error in the pressure vs. plastic strain curves is attained in 400 epochs. For the porosity vs. pressure curves, the best fit is obtained in 100 epochs and further improvement is not observed as the number of epochs is increased. Better fits may be obtained as the number of epochs increases if the data are shuffled during each epoch. Sigmoidal activation functions have been observed to produce worse fits to the data.

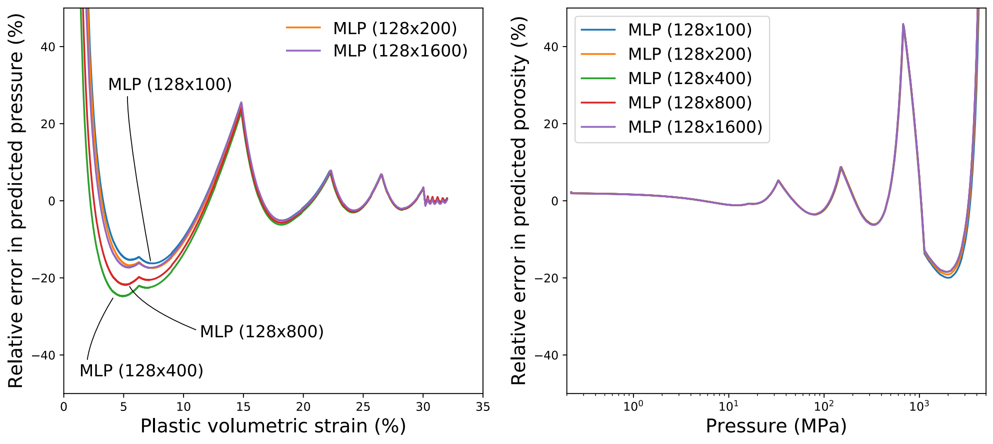


Figure 6 – *Relative prediction error for one-layer neural network fits to the crush-curve for a batch size of 128 and various training epochs.*

4.2 Two-layer model

Since the model with a single hidden layer does reproduce the crush-curve data accurately, it is worth considering a two-layer model. Let us design a model with 32 neurons in the first hidden layer and 64 neurons in the second hidden layer. Both layers use the ReLU activation function. The network is initialized with uniform random numbers. Random number generator seeds are identical to those used for the one-layer model. The plots in Figure 7 show the experimental data overlaid with model predictions from the two-layer neural net for a batch size of 128 and 800 training epochs. The predictions are visually quite close to the experimental data.

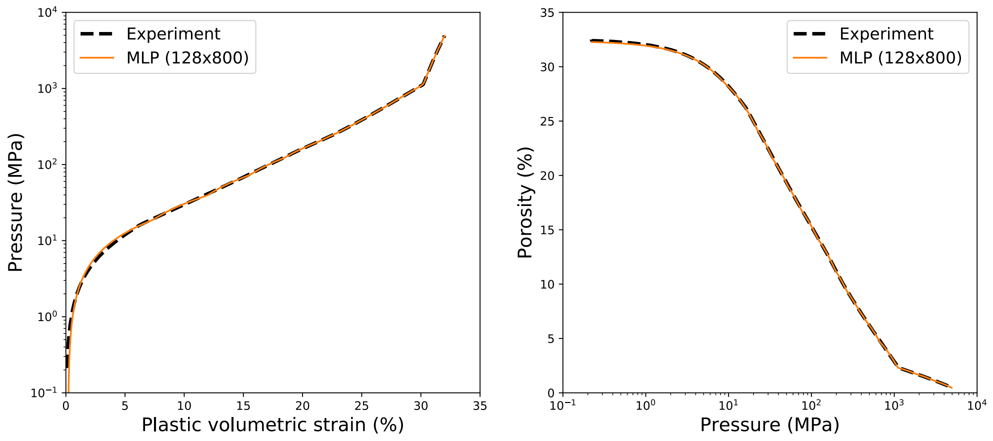


Figure 7 – *Two-layer neural network fits to the crush-curve for poorly-graded concrete sand.*

The relative prediction error for 800 training epochs and various batch sizes are shown in Figure 8. Statistics extracted from the absolute values of the relative error are presented in Table 1. The errors are smaller in magnitude for the two-layer model compared to the single layer neural network. It is instructive to observe the 75 percentile error given in the table. For both the pressure and the porosity model, a batch size of 64 produces the least error. However, the mean error is lowest for a batch of 128 as is the standard deviation.

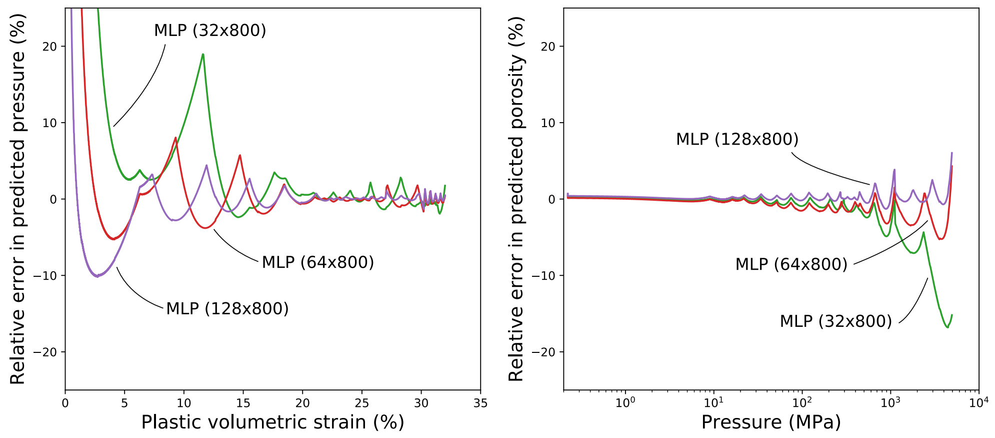


Figure 8 – *Relative error in predicted vs. experimental values for neural network fits to the crush-curve for varying batch sizes and 800 training epochs.*

Table 1 – *Prediction error statistics for concrete sand crush-curve data modeled with a two-layer network trained for 800 epochs.*

Batch size	16	32	64	128	256
		Absolute relative error (pressure)			
Mean	-	24.018766	11.757296	5.676102	8.937993
Std. dev.	-	54.073757	37.683897	19.975843	32.369637
Min.	-	0.008027	0.000001	0.000411	0.000232
25%	-	0.809280	0.443495	0.297135	0.247213
50%	-	2.772000	1.623539	1.423406	1.088924
75%	-	11.467700	4.400210	5.743104	3.109625
Max.	-	200.000000	200.000000	200.000000	200.000000
		Absolute relative error (porosity)			
Mean	0.962467	1.367971	0.850818	0.271043	-
Std. dev.	1.469681	3.046689	0.959453	0.459972	-
Min.	0.003487	0.000037	0.002541	0.000553	-
25%	0.317387	0.119464	0.224664	0.057574	-
50%	0.509094	0.248237	0.433117	0.154398	-
75%	0.951461	0.913366	1.225882	0.302870	-
Max.	11.948615	16.814478	5.283357	6.035958	-

Training time is not a limiting factor for small data sets such as those for the crush curve. However, the number of epochs can make a difference in the accuracy of the fit. This can be observed in Figure 9, where the predicted pressure settles to a relatively steady value only after 400 epochs. Similar behavior is observed for the predicted porosity curves.

From these experiments, it can be seen that the process of fitting a neural network model to a set of data requires a design of experiments grid to be explored, even for the simplest one-dimensional

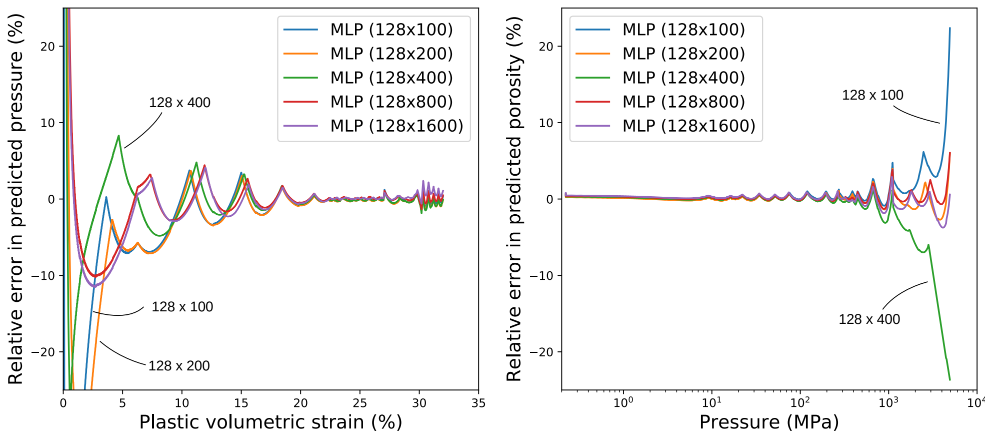


Figure 9 – *Relative error in predicted vs. experimental values for neural network fits to the crush-curve for a batch size of 128 and various training epochs.*

case. The selection of the most appropriate model can be facilitated by exmining error statistics systematically. For instance, a metric that selects the model with the least mean, standard deviation, minimum, and fourth-quartile error can be designed and used to determine an optimal model. Overfitting is not a significant issue for the crush-curve model because physical considerations can be used to control the behavior of the model beyond the range where it has been trained.

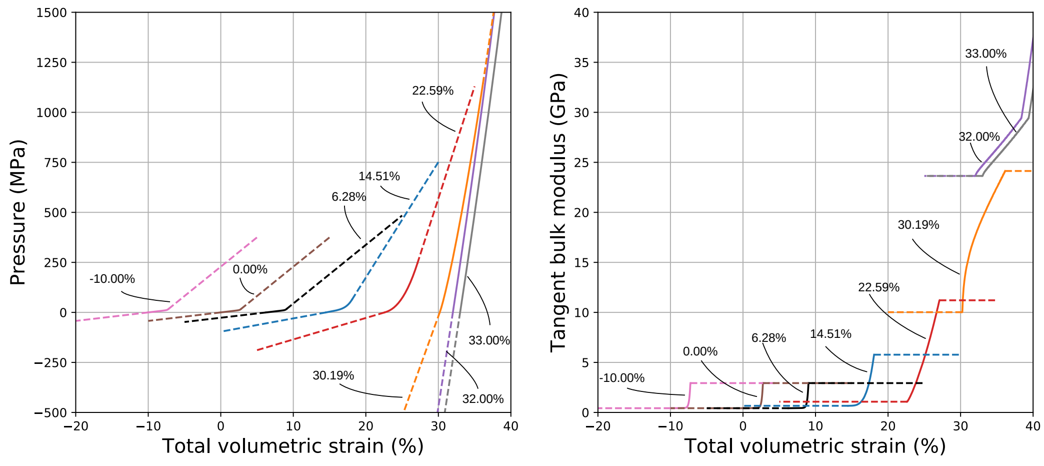
5 Bulk modulus model

A bulk modulus model can either be fitted directly to bulk modulus data (such as those in Figure 2(b)) or derived indirectly from pressure vs. strain data shown in Figure 2(a). The latter may be preferable for plasticity models where the stress integration algorithm uses $K = K(I_1, \varepsilon_v^p)$ while the former is more convenient when $K = K(\varepsilon_v^e, \varepsilon_v^p)$ where ε_p^e is the elastic volumetric strain.

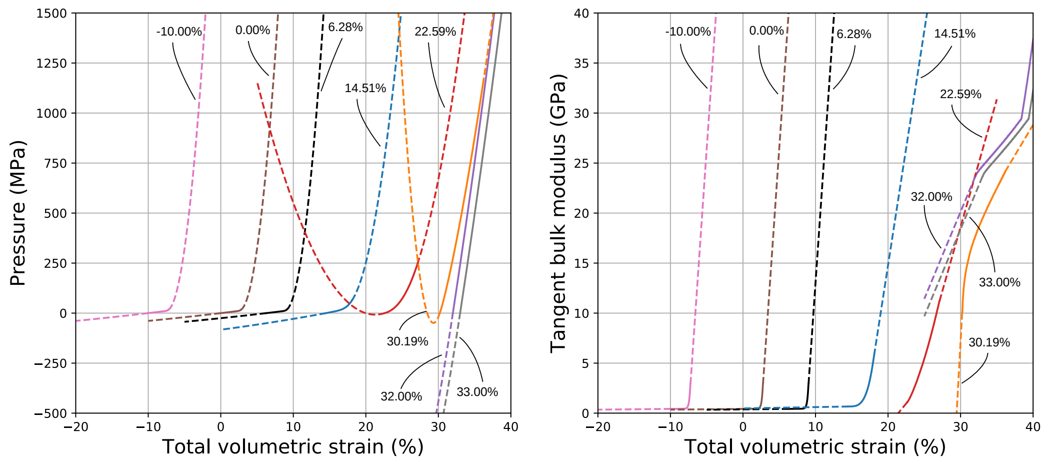
In Banerjee, Fox, and Regueiro (2020) we explored the possibility of using data of the form $p = p((\varepsilon_v^e, \varepsilon_v^p)$ as input to the fitting process and discovered that the constraints of zero pressure at zero volumetric elastic strain cannot be readily applied during the optimization process. The same situation arises when using MLP neural nets. Therefore, the model development process discussed in this paper uses input data of the form $p = p(\varepsilon_v, \varepsilon_v^p)$ where the total volumetric plastic strain is given by $\varepsilon_v = \varepsilon_v^e + \varepsilon_v^p$. Multilayer perceptron neural nets can be also used to directly fit data of the form $K = K(\varepsilon_v^e, \varepsilon_v^p)$ and we discuss such models too.

We have also found that bootstrapping (Mooney, Duval, and Duvall, 1993) the input data improves the quality of the fit, particularly for the low strain regime where fewer samples are available along a given unloading path. Before bootstrapping the data, a linear interpolation-based smoothing operation is used to sample each unloading curve at regular intervals. The input data where then extended in the tension and compression regimes and assumed curves were added to estimate behavior outside the range of the experimental data. These are required to avoid arbitrary behavior from the fitted neural net models for regimes where experimental data are not available.

The extended and padded input data are given in Figures 10(a) and (b). The curves in part (a) of the figure, shown the modified input data when the bulk moduli are assumed to be constant in the regions where the data have been extended. On the other hand, if we linearly extrapolate the bulk modulus and compute the corresponding pressures by integration, we get the data shown in part (b) of the figure. Note that bulk moduli can become zero and negative if extrapolated linearly. However, the elastic strain at which such behavior occurs is tensile and the neural network model is not applicable in that regime.



(a) Assuming constant bulk modulus.



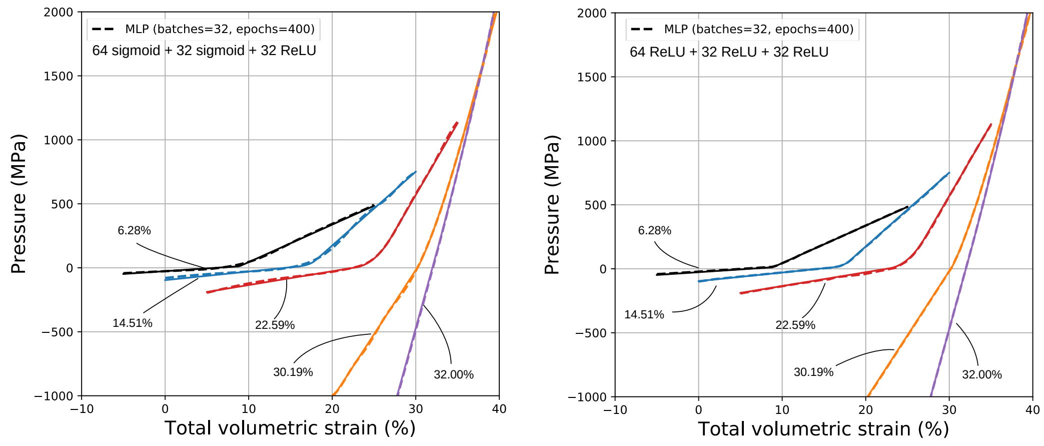
(b) Assuming linear bulk modulus.

Figure 10 – *Extended and padded elastic unloading curves and bulk moduli for dry poorly graded concrete sand. The solid lines show the original data. The dashed lines are the extended data. Percent volumetric plastic strains corresponding to each curve are also shown.*

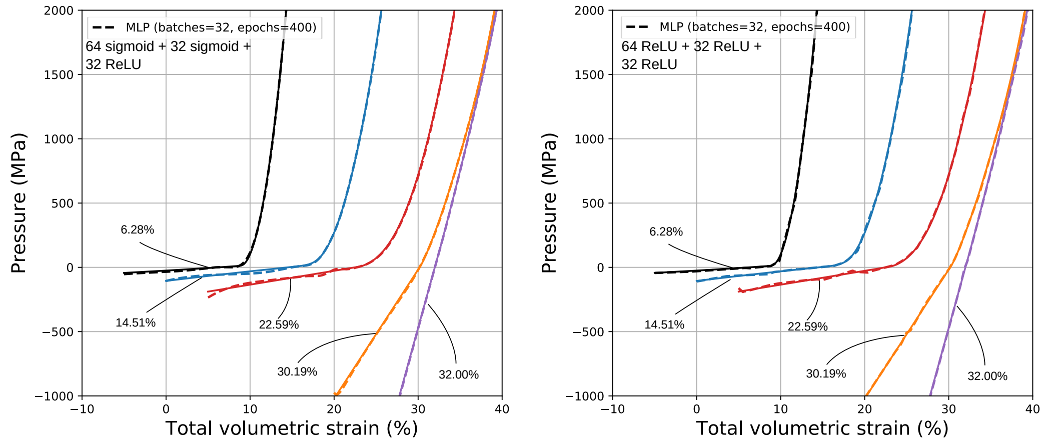
5.1 Fits to pressure-volumetric strain data

Neural networks topologies that can be evaluated rapidly are essential if such models are to be used in computational mechanics applications. We have discovered that at three-layer networks are the minimum depth at which the input pressure-strain data can be reproduced reasonably well. However, MLP models fail to generalize well for batch sizes larger than 32.

Visual examination of the MLP fits to the extended unloading curves in Figures 11(a) and (b) shows that the fit appears excellent. In fact, the absolute relative error between the experimental and predicted values of pressure is less than 20% at almost all points along each curve. Though slight perturbations are observed in the data extended assuming a linear bulk modulus, these do not appear significant.



(a) Constant bulk modulus extended curves.



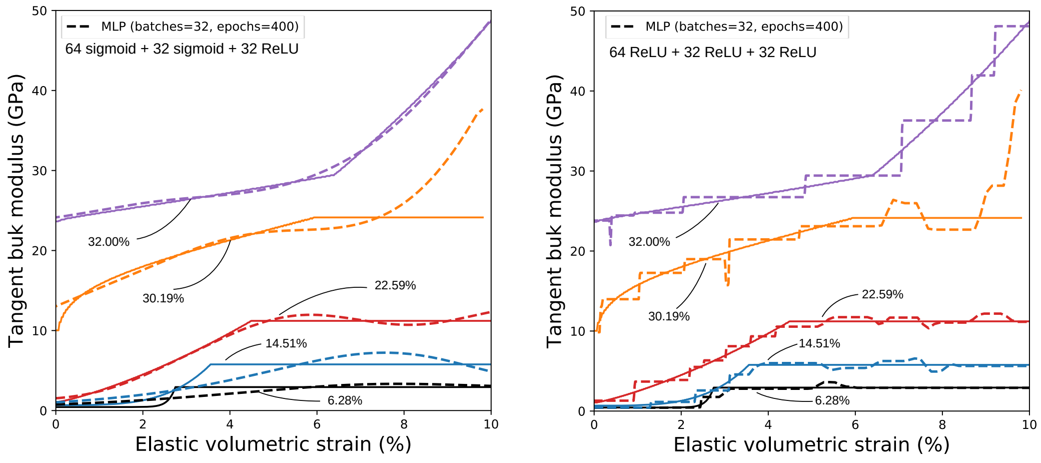
(b) Linear bulk modulus extended curves.

Figure 11 – *Three-layer MLP fits to pressure vs. total volumetric strain curves for concrete sand. The models on the left used sigmoid and ReLU activations. Those on the right used only ReLU activations.*

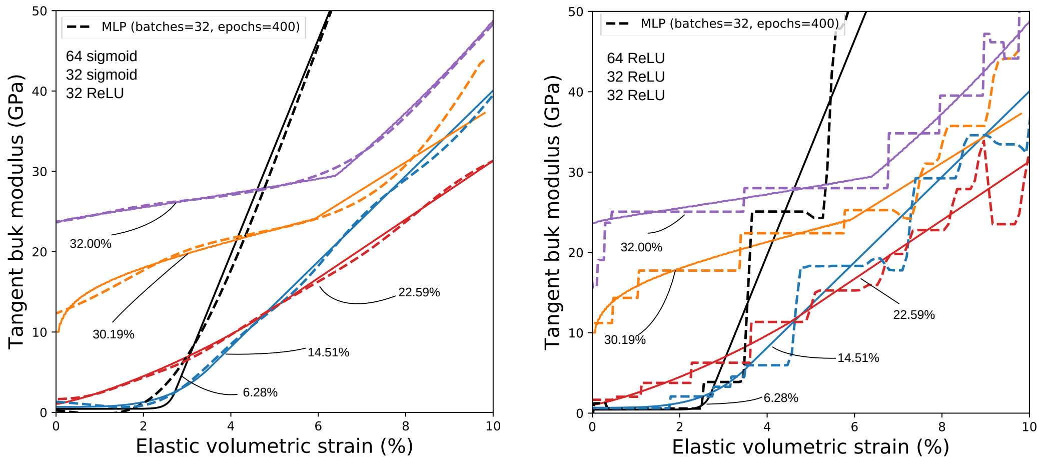
Note that the plots on the left use a topology containing 64 neurons in the first layer and sigmoid activation, 32 with sigmoid activation in the second layer, and 32 with ReLU activation in the third layer. The plots on the right are for the same topology but ReLU activation for all three layers.

If we examine the derivatives of these curves in the form of the tangent bulk modulus shown in

Figure 12(a) and (b), we observe remarkable differences between the two fits. The sigmoid-based models are smoother but tend to average out the sharp changes in gradient. However, the purely ReLU-based model appears to have produced a piecewise linear fit to the pressure-strain data that is reflected as stepped bulk modulus values.



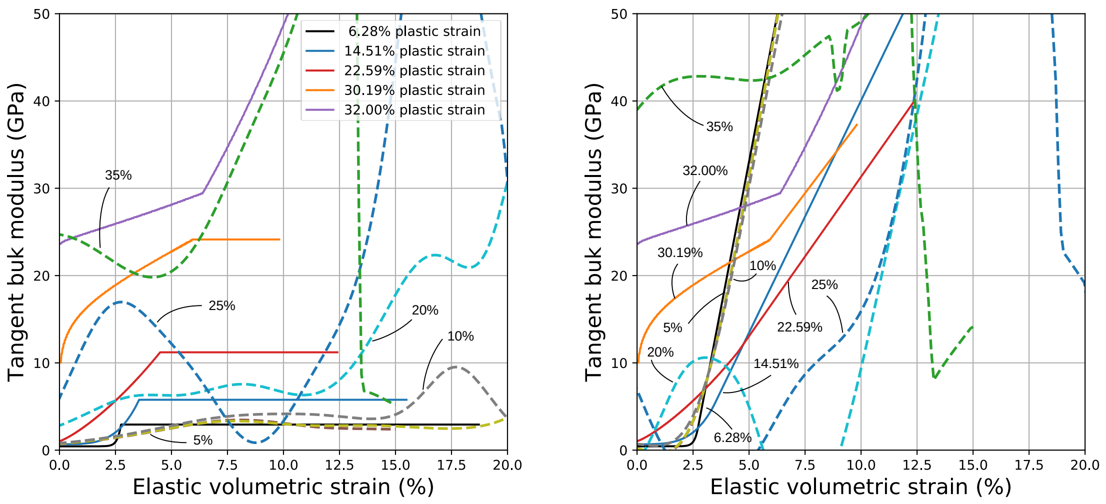
(a) Constant bulk modulus extended curves.



(b) Linear bulk modulus extended curves.

Figure 12 – *Bulk moduli computed from three-layer MLP to elastic unloading curves for concrete sand. The purely ReLU-based models are on the right. The models on the left used a combination of sigmoids and ReLU.*

When the MLP models with sigmoid activation are used to predict elastic unloading curves and then used to compute the bulk modulus for plastic strains outside the training set, we observe the behavior shown in Figures 13(a) and (b). If the model was obtained from fits to data extended assuming constant bulk moduli, the moduli for 5% and 10% plastic strain are reasonable. On the other hand, the modulus for 20% plastic strain is larger than that for 22.6% plastic strain up to an elastic strain of 2.5%. At 25% plastic strain, the predicted modulus is reasonable at strains below 2.5%, decreases to almost the initial value at 0% plastic strain, before increasing rapidly after around 8.5% strain. A similar trend is observed at 35% plastic strain. For the model fit to the linear bulk modulus extended data, the bulk modulus turns negative for several volumetric plastic strain values.



(a) Constant bulk modulus extended curves.

(b) Linear bulk modulus extended curves.

Figure 13 – *Predicted behavior of bulk modulus from MLP models of concrete sand for volumetric plastic strains outside the training set. Text labels indicate volumetric plastic strains. Both sets of data were modeled with MLP networks containing a combination of sigmoid and ReLU activation functions.*

The observations in this section suggest that fits to data of pressure vs. strain form may not be appropriate for MLP models because the derivatives need to be calculated to determine the bulk modulus. Similar results have been obtained for a range of batch and epoch sizes, various network topologies, and different types of scaling of the input data. However, the accurate fits to the input data indicate that direct fits to bulk modulus curves may be easier to generalize.

5.2 Fits to bulk modulus-volumetric strain data

Instead of computing derivatives as a secondary step after fitting pressure-strain data, it is more convenient to fit MLP models directly to bulk modulus-strain data. We focus on the data extended with a constant bulk modulus assumption as seen in Figure 13(a). Notice from Figure 13(b) that multiple values of bulk modulus occur at the same volumetric plastic strain in the experimental data extended using a linear bulk modulus assumption, causing the model to fail to generalize adequately. This issue is less relevant for the data in Figure 13(a).

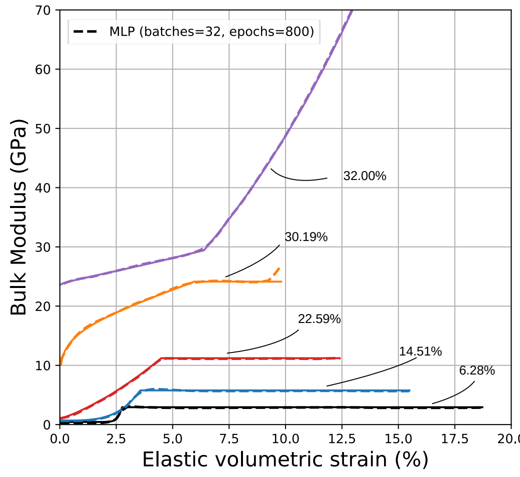
The bulk moduli were computed from smoothed and resampled pressure-volumetric strain curves extended in tension and compression. Extra data at plastic strains of -10%, 0%, and 33% were added to the data set as shown in Figure 10. The data were were then translated to have zero mean and scaled such the standard deviation was 1. After that the scaled data were bootstrapped such that the bulk modulus curves at all values of volumetric plastic strain had an approximately equal number of samples. The procedure discussed earlier was used to fit neural network models to these data.

Models produced by the following three dense three-layer neural network topologies are discussed:

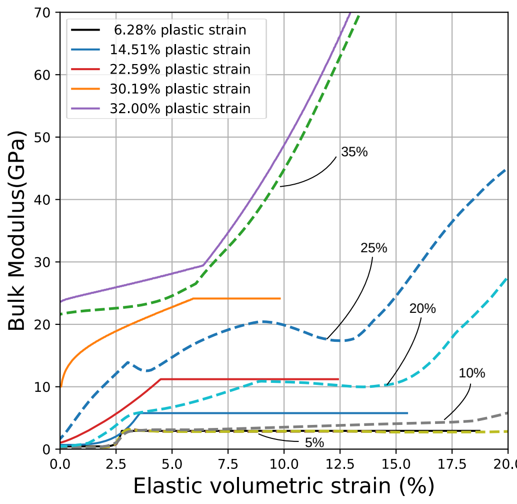
1. Model A: 64 neurons with sigmoid activation in layer 1, 32 neurons with sigmoid activation in layer 2, 32 neurons with ReLU activation in layer 3.
2. Model B: 64 neurons with ReLU activation in layer 1, 32 neurons with ReLU activation in layer 2, 32 neurons with ReLU activation in layer 3.
3. Model C: 64 neurons with sigmoid activation in layer 1, 32 neurons with sigmoid activation in layer 2, 32 neurons with sigmoid activation in layer 3.

Note that Model B is the least computationally expensive to evaluate while Model C is the most expensive due to the need to evaluate exponentials. Model A has an intermediate cost.

Fits to the bulk modulus data produced by Model A are shown in Figure 14 (a). The model reproduces the input data remarkably well and is a significant improvement over support vector regression (see Banerjee, Fox, and Regueiro (2020)). We can observe the generalizability of the model in Figure 14 (b). The model performs well for volumetric plastic strains below 14% but produces inaccurate results at larger plastic strains (when compared with linear interpolation between the input curves).



(a) Fits to input data.



(b) Predicted curves.

Figure 14 – *Model A: Predicted behavior of bulk modulus from three-layer MLP models of concrete sand. The model used 64 sigmoid neurons in the first layer, 32 sigmoid neurons in the second, and 32 ReLU neurons in the third. Text labels indicate volumetric plastic strains.*

The ReLU-based Model B produces a piecewise linear approximation to the input curves and is also quite accurate (Figure 15(a)). As can be seen from Figure 15(b), the quality of generalization is better than that produced by Model A.

In Figure 16 we see the fits to the data produced by the sigmoid-based Model C. The fits are again quite accurate and the model also generalizes reasonably well - better than Model A but worse than

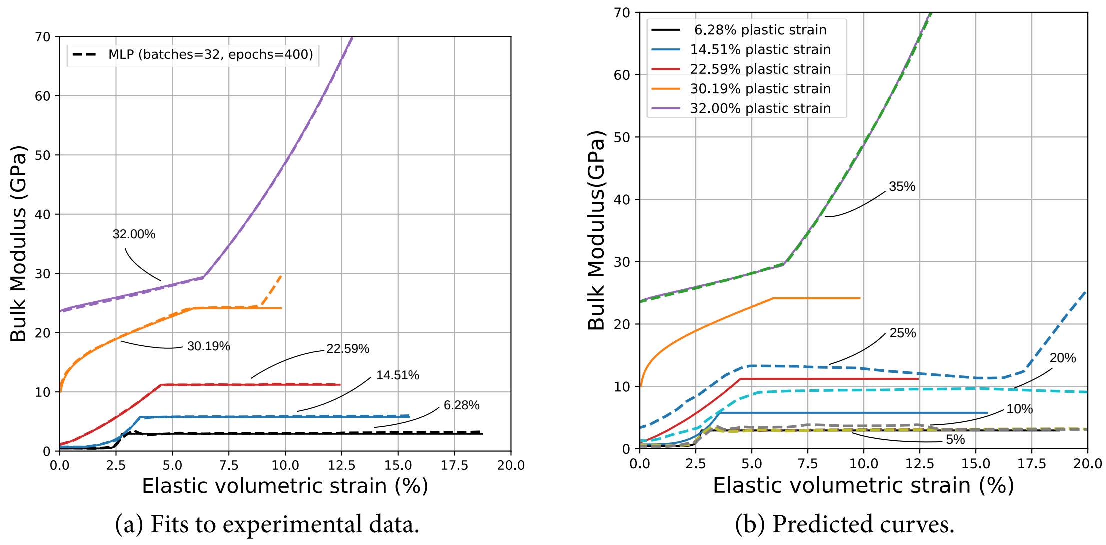


Figure 15 – *Model B: Predicted behavior of bulk modulus from MLP models of concrete sand for volumetric plastic strains outside the training set. Text labels indicate volumetric plastic strains.*

model B.

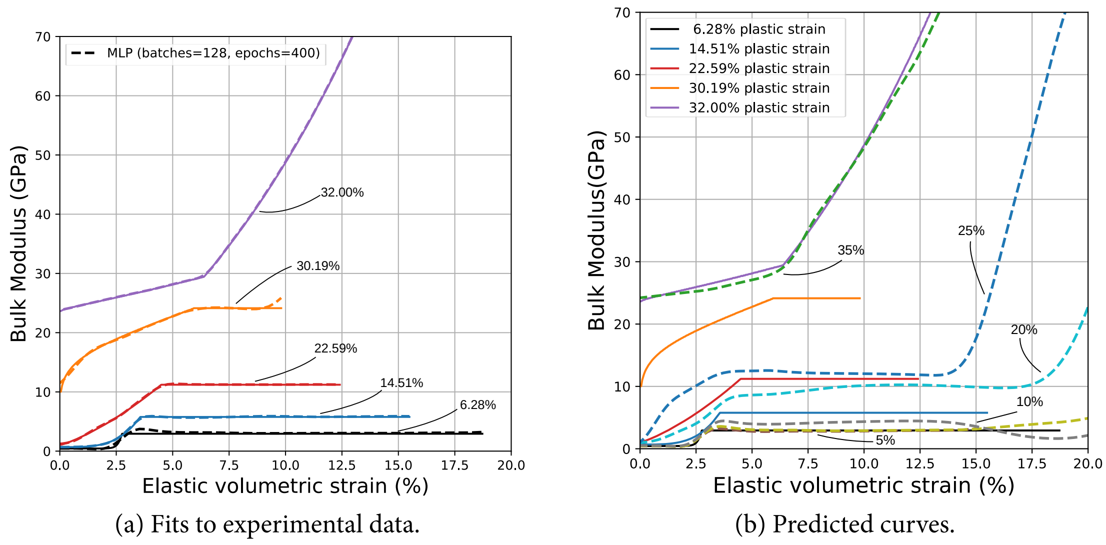


Figure 16 – *Model C: Predicted behavior of bulk modulus from MLP models of concrete sand for volumetric plastic strains outside the training set. Text labels indicate volumetric plastic strains.*

The results in this section suggest that ReLU-dominant neural networks can fit data and also generalize reasonably well. Of course, the choice of topology depends strongly on the type of data being modeled. For the bulk modulus data discussed in this paper, a purely ReLU-based model is significantly more computationally efficient while producing reasonable generalization to plastic strains outside the training set.

6 Concluding remarks

We have discussed the process of designing and fitting multilayer perceptron neural networks for modeling material models that involve multiple independent variables. Even when there is only a single independent variable, we find that at least two layers are need to fit the input data within a few percent relative error. In general, the larger the number of independent variables, the larger the number of layers needed to fit the input data. Clearly, the number of possible topologies increases exponentially as the number of layers increases. Until an automated search of the design space can be performed at reasonable cost, a design of experiments combined with best practices is currently the most feasible way to design topologies.

From a computational mechanics point of view, we would like to minimize the cost of evaluation of material models. The ReLU function is the computationally least expensive activation function that produces a good fit to the data and also generalizes well. As has become best practice in the machine learning world, we also suggest the use of this activation function. However, the piecewise linear fits produced by a ReLU-based model may not be appropriate if derivatives of the model are to be computed. In such situations we suggest fitting the neural network model to the derivatives and integrating to recover the original signal.

Acknowledgements

This research has been partially funded by the US Office of Naval Research PTE Federal award number N00014-17-1-2704.

References

Abadi, Martín et al. (2016). "Tensorflow: A system for large-scale machine learning". In: *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pp. 265–283 (cit. on pp. 4, 6).

Banerjee, B. and R. M. Brannon (2017). *Theory, verification, and validation of the ARENA constitutive model for applications to high-rate loading of fully or partially saturated granular media*. Tech. rep. PAR-10021867-1516.v1. Parresia Research Limited and University of Utah. DOI: 10.13140/RG.2.2.10671.53922 (cit. on p. 2).

— (2019). "Continuum modeling of partially saturated soils". In: *Shock Phenomena in Granular and Porous Materials*. Ed. by T. Vogler and A. Fredenburg. Springer (cit. on p. 2).

Banerjee, B., D. M. Fox, and R. A. Regueiro (2020). *Support vector regression for fitting multi-variable material models*. Tech. rep. PAR-10021867-092020-1. Parresia Research Limited (cit. on pp. 2, 4, 6, 10, 15).

Brannon, R. M. et al. (2015). "KAYENTA: Theory and User's Guide". In: *Sandia National Laboratories report SAND2015-0803* (cit. on pp. 2, 4).

Fox, D. M. et al. (2014). "The effects of air filled voids and water content on the momentum transferred from a shallow buried explosive to a rigid target". In: *International Journal of Impact Engineering* 69, pp. 182–193 (cit. on p. 4).

Gulli, Antonio and Sujit Pal (2017). *Deep Learning with Keras*. Packt Publishing Ltd (cit. on pp. 4, 6).

Hahnloser, Richard HR et al. (2000). "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit". In: *Nature* 405.6789, p. 947 (cit. on p. 3).

Harris, Charles R. et al. (Sept. 2020). "Array programming with NumPy". In: *Nature* 585.7825, pp. 357–362. ISSN: 1476-4687. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2 (cit. on p. 6).

Haykin, Simon (1994). *Neural networks: a comprehensive foundation*. Prentice Hall PTR (cit. on p. 2).

McKinney, Wes (2011). "pandas: a foundational Python library for data analysis and statistics". In: *Python for High Performance and Scientific Computing* 14 (cit. on p. 6).

Mooney, Christopher Z, Robert D Duval, and Robert Duvall (1993). *Bootstrapping: A nonparametric approach to statistical inference*. 94-95. Sage (cit. on p. 10).

Pedregosa, F. et al. (2011). "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830 (cit. on p. 6).

Rall, Louis B and George F Corliss (1996). "An introduction to automatic differentiation". In: *Computational Differentiation: Techniques, Applications, and Tools* 89 (cit. on p. 4).

Rumelhart, David E, Geoffrey E Hinton, and Ronald J Williams (1985). *Learning internal representations by error propagation*. Tech. rep. California Univ San Diego La Jolla Inst for Cognitive Science (cit. on p. 2).