

# Use of Machine Learning Techniques to Estimate System Performance

**James Randall Way**  
Aberdeen Proving Ground, MD  
UNITED STATES OF AMERICA  
[james.r.way14.civ@army.mil](mailto:james.r.way14.civ@army.mil)

## ***ABSTRACT***

*The Infantry Warrior Simulation (IWARS) is an entity-level combat simulation that is typically used to estimate differences in operational effectiveness caused by using different equipment, including grenades and grenade launchers. When a simulated grenade explodes in IWARS, the effect on nearby personnel is determined by looking up a probability of incapacitation value that was precomputed by a high-resolution model. This value depends on many factors, creating the need for a large lookup table that may exceed the maximum database size. To solve this problem, a neural network input option was created, giving analysts the opportunity to use highly compressed data without sacrificing accuracy or runtime. Previous compression techniques are either less accurate or offer a lower compression ratio.*

*This research was performed in fiscal year 2019 as part of the study titled “Machine Learning Techniques to Aid in Generating Item-Level Performance Estimates for use in Squad and Soldier Level Operational Assessments”. The other half of that study is discussed in a separate report. In that half of the study, gradient-boosted decision trees were used to successfully predict the surrogation decisions of human subject-matter experts (SMEs). (When data is not available for a requested system, a similar system is often used as a surrogate.) The trained decision tree models can be used to suggest surrogates for future data requests, reducing the time needed to fulfil those requests and improving the accuracy of the provided data.*

Keywords: Computers-computer science: Artificial Intelligence; neural networks; gradient-boosted decision trees

## **1. INTRODUCTION**

### **1.1 Background**

The Infantry Warrior Simulation (IWARS) is an entity-level combat simulation focused on the dismounted soldier, squad, and platoon that is typically used by the Army to estimate differences in operational effectiveness caused by using different items of equipment. In particular, IWARS has been used to compare the effectiveness of different grenades and grenade launchers [1, 2, 3], helping to guide the development and acquisition of those systems.

### **1.2 Problem Statement**

When a simulated grenade explodes in IWARS, the effect on nearby personnel is determined by looking up a probability of incapacitation (P(I)) value that was precomputed by a high-resolution model. The P(I) value depends on many factors, including the target's posture, body armor, and mission (attack or defend), as well as the munition angle of fall, burst height, burst-to-target range, burst-to-target azimuth angle, and post-wounding time. The P(I) lookup table can be very large due to this large number of factors. In fact, a high-resolution lookup table is often too large to fit into the maximum IWARS database size of about 150 megabytes.

To solve this problem, analysts may split the IWARS database into smaller parts. For instance, an analysis of 12 new types of air-bursting grenade could be performed by creating 12 IWARS databases, one per grenade. This strategy will fail if the lethality data describing one type of grenade is too large, or if more than one type of grenade is required in a particular scenario but only one grenade's lethality data will fit into a single database. Moreover, even if this strategy is possible, there are drawbacks: any additional database changes would have to be mirrored 12 times, and the size of the databases would slow down IWARS and the database editor tool.

Alternatively, analysts may circumvent the database size limitation by using lower-resolution P(I) data. This is often done by deleting certain burst heights and burst-to-target ranges, and averaging together P(I) values for groups of burst-to-target azimuth angles. This reduces the accuracy of the simulation and reduces confidence in the results.

### **1.3 Purpose**

The purpose of this paper is to document a new solution to this problem, one that works in all cases and with virtually no loss in accuracy or increase in model runtime. It can be described as follows:

1. Train artificial neural networks to learn the P(I) values. The neural network parameter values will then encode the original P(I) data, thereby compressing it [4].
2. Recreate those neural networks within IWARS in order to estimate P(I) values as they are needed [5].

## **2. COMPRESSION TECHNIQUE USING ARTIFICIAL NEURAL NETWORKS**

### **2.1 Methodology**

An artificial neural network, hereafter referred to simply as a neural network, is a computer system that is inspired by how a brain processes information. It is composed of highly interconnected processing elements, or neurons, working in unison to solve a specific classification or regression problem. The organization and weights of the connections between the neurons determines the output. A neural network learns by example. Learning involves adjustments to the connections that exist between the neurons to reduce predictive error.

The most important characteristic of neural networks in this application is that they are equivalent to sophisticated regression equations. Figure 1 shows both the graphical and algebraic representations of a neural network regression equation [6, p. 1.14]. In this example, the inputs consist of the firer-to-target range, burst height, azimuth angle, burst-to-target range, and casualty criterion index. Those inputs feed into the artificial neurons in the first layer, each of which computes a weighted sum of the inputs and the value of some nonlinear "transfer" function of that weighted sum. The output of the first layer is then used as input to the second layer, which then feeds into the third layer. The third layer then produces the predicted output, which in this example is a probability of incapacitation.

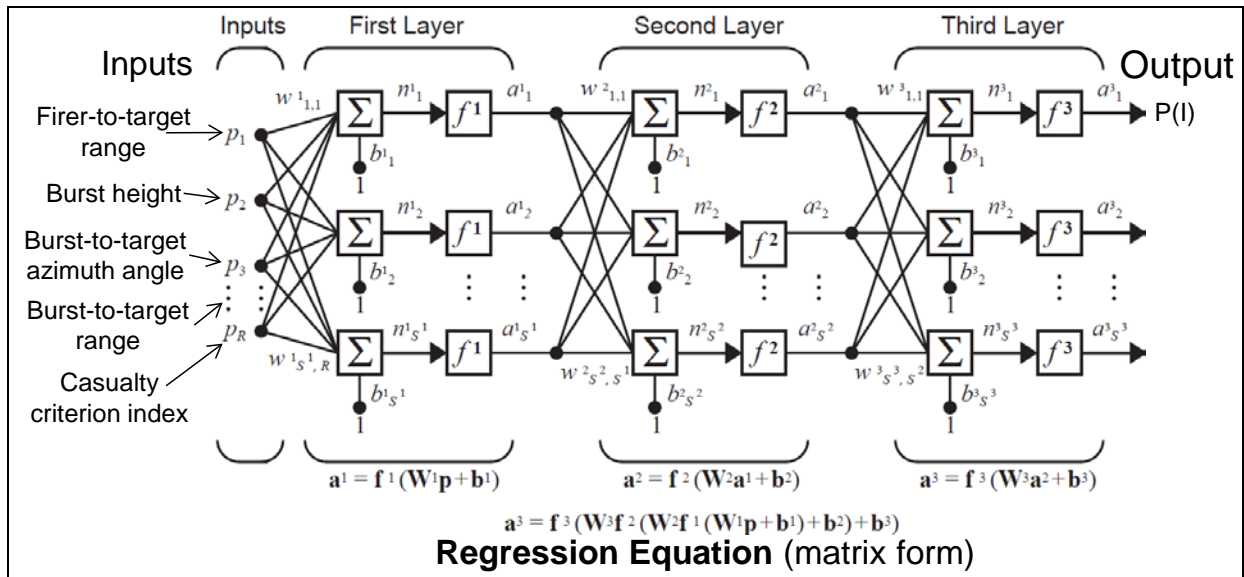


Figure 1: Graphical and algebraic representations of an artificial neural network with three hidden layers.

The process of adjusting the weights of a neural network so that the predicted output better matches example output is called training. Training algorithms can be quite sophisticated, but they all have a similar structure:

1. Inputs are fed into the neural network and output is calculated.
2. The predicted output is compared to the known output to calculate the error.
3. Weights are adjusted to decrease the error.
4. Training ends when the error is reduced below some threshold, or after the weights stop changing very much, or after a certain number of iterations.

The neural network compression technique can be summarized as follows: An artificial neural network is a regression equation with many free parameters that can, in theory, approximate any reasonable function arbitrarily well [6, p. 1.15]. If the number of parameters is less than the size of the training data, the result is data compression. In this application the output of the neural network is  $P(I)$ , and the inputs are firer-to-target range, burst height, burst-to-target azimuth angle, burst-to-target range, and casualty criterion index. The goal is to create and train the smallest neural network that can learn the  $P(I)$  values with acceptable accuracy.

Training a neural network is computationally expensive. The results shown in this paper were achieved using a computer with 256 gigabytes of memory, one NVIDIA Tesla K40 graphics processing unit (GPU), and 40 central processing unit (CPU) cores provided by two Intel Xeon E5-2698v4 CPUs

## 2.2 Preliminary Results

Preliminary work using neural networks to compress  $P(I)$  data showed that compression ratios of over 1000:1 are possible with little loss of accuracy [4], where compression ratio is defined in Equation (1). Some manual trial and error was done to optimize the neural network parameters, including the number of layers and the number of neurons per layer.

$$\text{Compression Ratio} = \frac{\text{Uncompressed Size}}{\text{Compressed Size}} \quad (1)$$

One technique used to check the accuracy of a neural network was to graph the original P(I) values next to those approximated by the network. The P(I) values were visualized using what is sometimes called a “bugsplat” or “butterfly chart”, an example of which is shown in Figure 2.

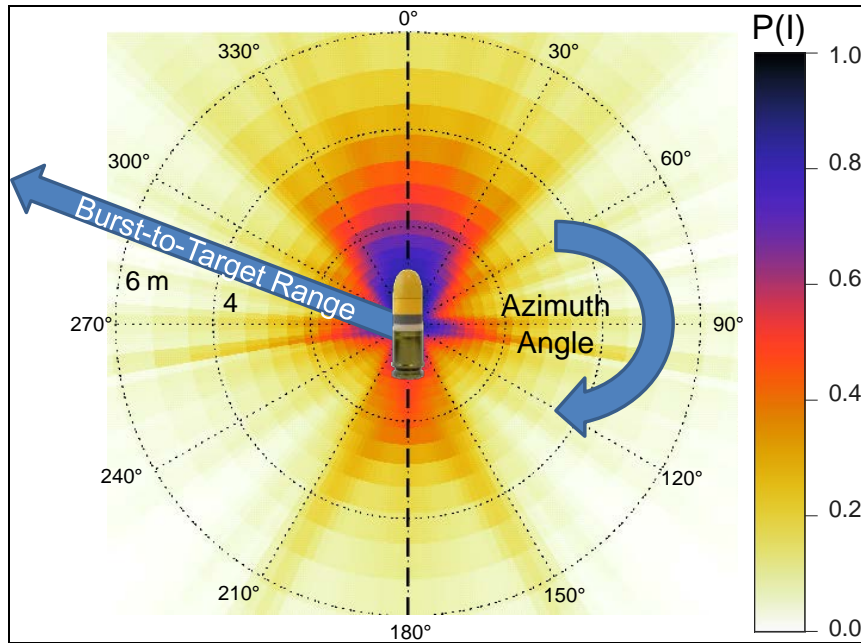


Figure 2: Visualization of P(I) data for one firer-to-target range and burst height.

An example comparison between original P(I) values and those approximated by a neural network is shown in Figure 3.

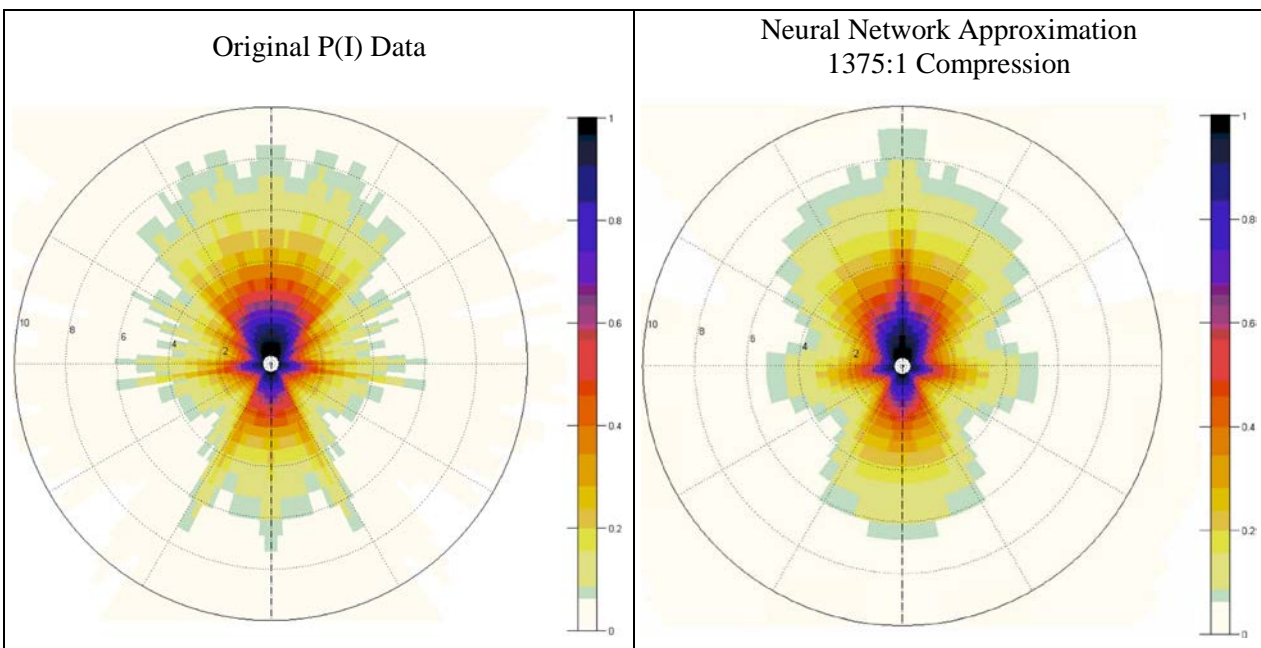
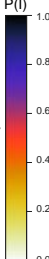
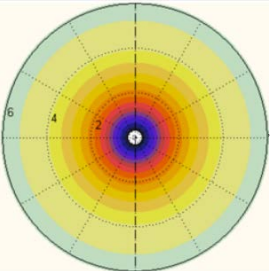
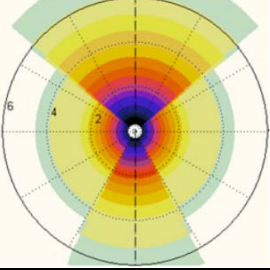
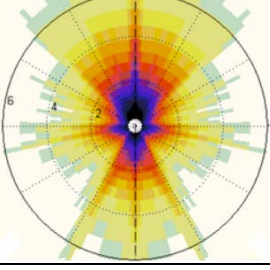
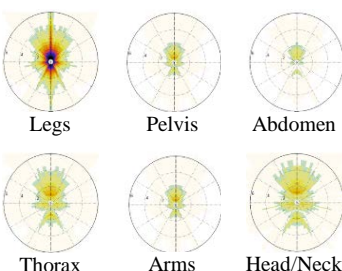


Figure 3: Original P(I) data (on the left) vs. neural network approximation (on the right).

It was shown in many examples that the current techniques used to compress P(I) data are either less accurate or have a lower compression ratio than the neural network method. The current P(I) compression techniques correspond to the first three data formats described in Table 1, namely the Carleton damage function, P(I) vs. range (PIVR), and P(I) vs. range and angle (PIVR\_Angle). The last two data formats in Table 1 are for uncompressed data. The same lethality data is shown in each of the different formats to provide a visual comparison.

Table 1: Current fragmenting munition lethality data formats.

Name	Description	Example
Carleton Damage Function	Assumes P(I) drops off exponentially: $P(I) = D_0 e^{-\pi D_0 r^2 / A_L}$ where $D_0$ is zero-range P(I), $A_L$ is lethal area, and $r$ is burst-to-target range.	
PIVR (P(I) vs. Range)	P(I) vs. burst-to-target range. Does not model azimuth angle.	
PIVR_Angle (P(I) vs. Range and Angle)	P(I) vs. range and 3 angular zones. Requires judgment in defining zones.	
Whole-Body Grid	P(I) vs. range and 91 azimuth angles.	
By-Body-Part Grid	P(I) vs. range, 91 azimuth angles, and 6 body-parts. Can be used to improve modeling of partial cover.	

The accuracy of the various compression techniques was evaluated not just by analyzing the differences between the original and compressed data, but by comparing the results of using the different datasets in the simple combat simulation known as FBAR [7], which is named after its primary output,  $\bar{F}$ , meaning Expected Fraction of Casualties (EFC). An EFC value of zero means that no targets are incapacitated in any trial, and an EFC value of one means that every target is incapacitated in all trials.

In one FBAR-based comparison, using Carleton, PIVR, and PIVR\_Angle input data all produced significantly different results from using the original P(I) data, which was a whole-body grid. The poor performance of these previous compression techniques can be attributed to the munition’s highly directional fragmentation pattern. In contrast, using P(I) data from a trained neural network produced results that matched the original P(I) results almost exactly, as shown in Figure 4. The EFC values in this figure have been normalized to be relative to the results from using the original P(I) data.

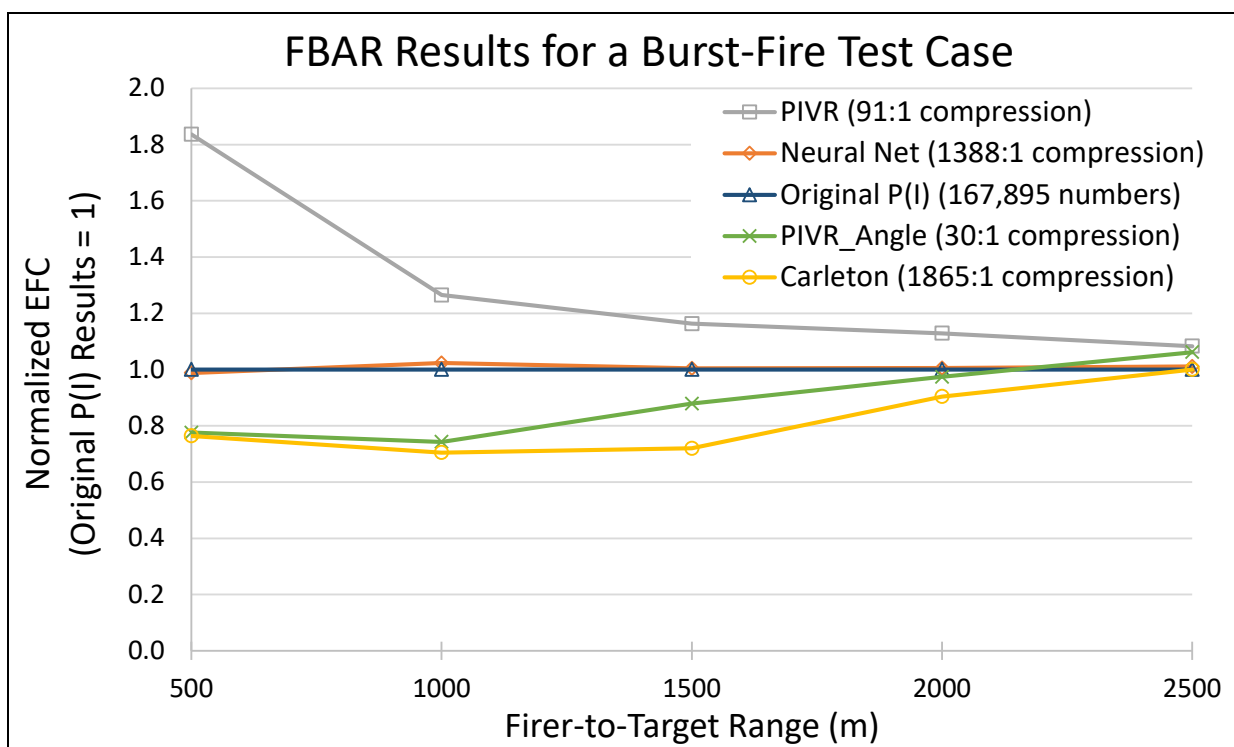


Figure 4: Normalized EFC as a function of range and lethality data format.

Neural networks were also shown to accurately predict missing data. In particular, a neural network that was trained on P(I) values for standing, prone, and foxhole target postures was able to accurately predict P(I) values for crouching and kneeling target postures. However, the ability to predict this missing data was only possible after adding some random noise to the target dimensions in the training data.

### 2.3 Bayesian Optimization

To train a neural network, one must specify the network architecture and various options for the training algorithm. Selecting and tuning these parameters manually can be difficult. Therefore, parameters were tuned using MATLAB’s Bayesian optimization algorithm, which can be used to optimize functions that are non-differentiable, discontinuous, and time-consuming to evaluate. The algorithm internally maintains a Gaussian process model of the function being optimized (the objective function), and uses objective function evaluations to train this model [8]. Bayesian optimization was used to estimate the network size that gave the

least error after a fixed amount of training time:

- The objective function was mean squared error (MSE) on the training set of 66,339 P(I) values for the rear-fragmenting munition described in section 3.2.
- The hidden layer transfer function was hyperbolic tangent ('tansig' in MATLAB).
- The variable being optimized was the number of neurons in each of two hidden layers.
- Thirty neural networks were trained for three minutes each using Levenberg-Marquardt backpropagation (MATLAB's 'trainlm' function) with default parameters. The default parameters included a maximum mu value of  $10^{10}$  and a minimum gradient of  $10^{-7}$ . In addition to the maximum time limit of three minutes, reaching the maximum mu value or minimum gradient would also cause the training to stop. However, that rarely happened.
- The minimum observed MSE of  $2.4 \times 10^{-5}$  occurred when there were 20 neurons per hidden layer, as shown in Figure 5. This figure also shows the total number of neural network weight and bias values, which increases quadratically with respect to the number of neurons per hidden layer. The compression ratio at the observed minimum was 118:1.

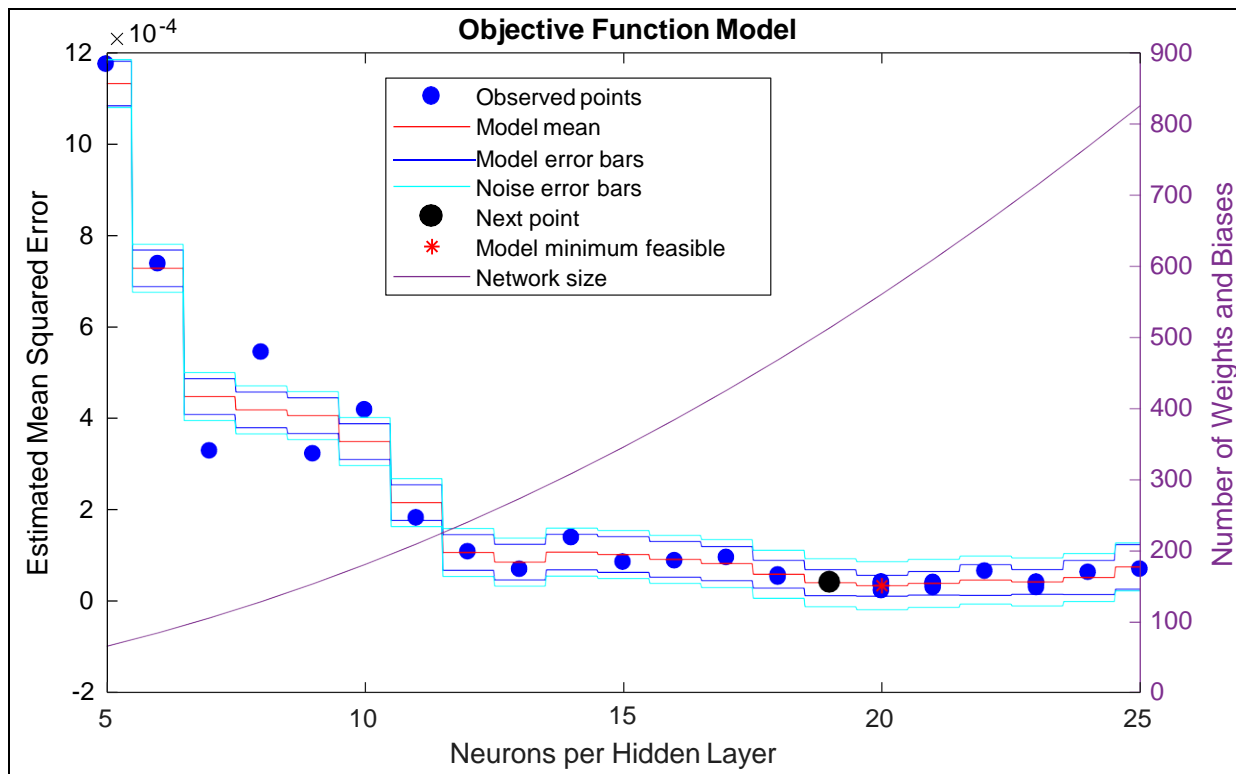


Figure 5: Bayesian optimization of network size.

Since the optimal network size may depend on the training method, another Bayesian optimization was performed in order to estimate which combination of training method and network size give the least error after a fixed amount of training time. In this case:

- The objective function was  $\log_{10}(\text{MSE})$  on the training set of 66,339 P(I) values for the rear-fragmenting munition described in section 3.2.
- The hidden layer transfer function was hyperbolic tangent.

- The variables being optimized were the training method and the number of neurons in each of the two hidden layers.
- Eighty neural networks were trained for three minutes each using a variety of training methods with default parameters. The GPU was used for training methods that could support it, which included all of the training functions except for trainlm and trainbr, as shown in Table 2.
- The minimum observed MSE of  $1.3 \times 10^{-5}$  occurred when there were 27 neurons per hidden layer (corresponding to a compression ratio of 70:1) and when the training method was Levenberg-Marquardt backpropagation (MATLAB's 'trainlm' function), as shown in Figure 6. The training functions are described in Table 2.

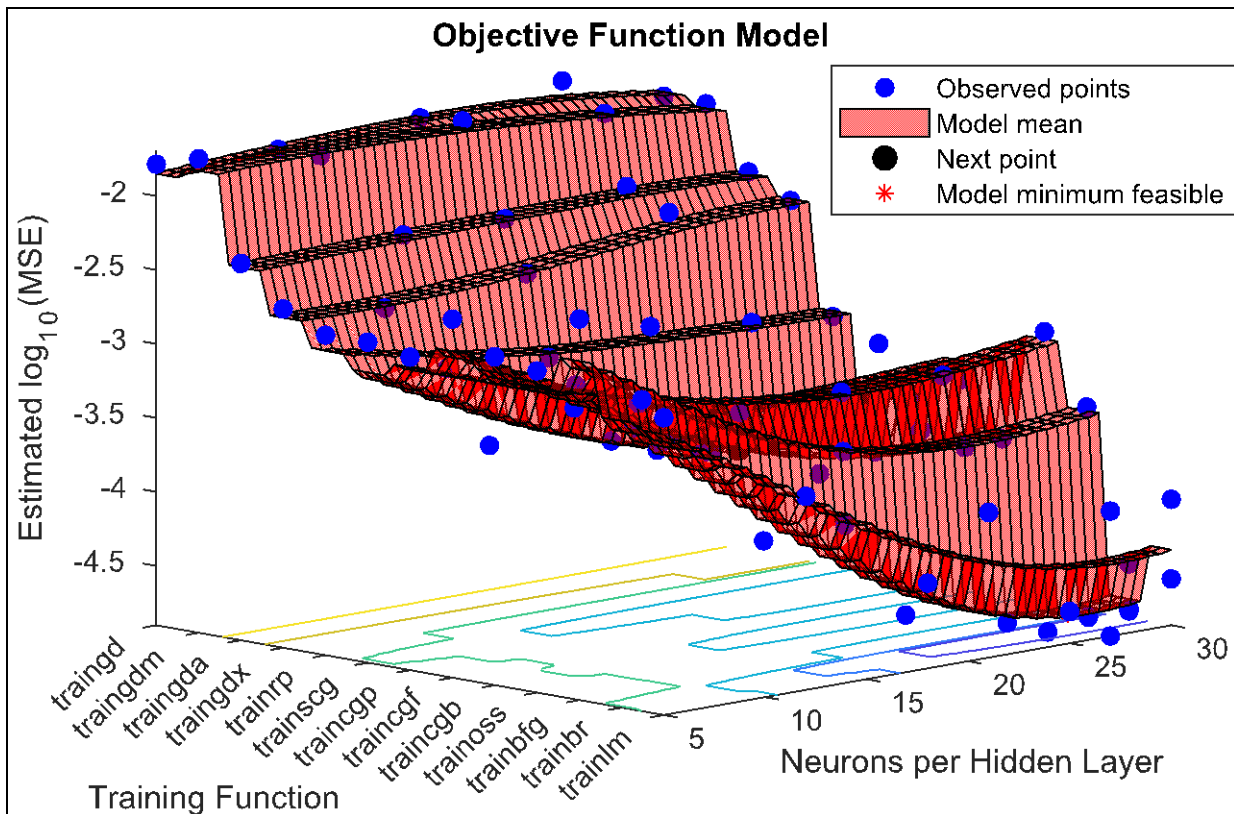


Figure 6: Bayesian optimization of network size and training function.



The MATLAB neural network training functions shown in Figure 6 are briefly described in Table 2.

**Table 2: MATLAB neural network training functions.**

Name	Description	GPU Support?
trainlm	Levenberg-Marquardt: Newton-like method that is often the fastest algorithm in the toolbox, although it does require more memory than other algorithms [9].	no
trainbr	Levenberg-Marquardt with Bayesian Regularization: Minimizes a combination of squared errors and weights to produce a network that generalizes well [10].	no
trainbfg	Broyden-Fletcher-Goldfarb-Shanno Quasi-Newton: Like all quasi-Newton methods, this method approximates the Hessian matrix (second derivatives of the objective function with respect to the network weights and biases) to generally converge in fewer iterations than conjugate gradient methods, although it does require more memory and computation per iteration [11].	yes
trainoss	One Step Secant: Can be considered a compromise between full quasi-Newton (secant) algorithms and conjugate gradient algorithms.	yes
traincgb	Conjugate Gradient with Beale-Powell Restarts: In contrast to the basic backpropagation algorithm that adjusts parameters in the steepest descent direction (negative of the gradient), conjugate gradient algorithms adjust parameters in a direction that is conjugate to adjustment directions of previous iterations [12, 13]. All conjugate gradient algorithms must reset the search direction to the negative of the gradient when the number of iterations is equal to the number of network parameters. However, the traincgb function also restarts if there is very little orthogonality left between the current gradient and the previous gradient [14].	yes
traincgf	Fletcher-Powell Conjugate Gradient: All conjugate gradient algorithms compute the next search direction as the sum of the negative of the gradient and some scalar times the previous search direction. For the traincgf algorithm, this scalar is equal to the ratio of the norm squared of the current gradient to the norm squared of the previous gradient [15].	yes
traincgp	Polak-Ribière Conjugate Gradient: All conjugate gradient algorithms compute the next search direction as the sum of negative of the gradient and some scalar times the previous search direction. For the traincgp algorithm, this scalar is equal to the inner product of the previous change in the gradient with the current gradient divided by the norm squared of the previous gradient [16].	yes
trainscg	Scaled Conjugate Gradient: Avoids the time-consuming line search used in other conjugate gradient methods for determining an appropriate step size [17].	yes
trainrp	Resilient Backpropagation: Uses only the sign of derivatives to avoid slow training caused by small derivative magnitudes [18].	yes
traingdx	Gradient descent with momentum and adaptive learning rate [19]	yes
traingda	Gradient descent with adaptive learning rate	yes
traingdm	Gradient descent with momentum	yes
traingd	Gradient descent backpropagation	yes

The previous optimizations were for networks with an equal number of neurons per hidden layer, which may not be the best distribution of neurons. Therefore, a Bayesian optimization was performed on the combination of hidden layer sizes. In this case:

- The objective function was mean squared error on the training set of 66,339 P(I) values for the rear-fragmenting munition described in section 3.2.
- The hidden layer transfer function was hyperbolic tangent.
- The variables being optimized were the number of neurons in the first hidden layer and the number of neurons in the second hidden layer.
- 180 neural networks were trained for two minutes each using Levenberg-Marquardt backpropagation with default parameters.
- Only networks with 3000 or fewer weight and bias values were considered, meaning that the compression ratios were at least 22:1.
- The minimum observed MSE of  $2.5 \times 10^{-5}$  occurred when there were 63 neurons in the first hidden layer and 2 neurons in the second hidden layer, as shown in Figure 7. The estimated minimum is far away from the observed minimum in this case, possibly because there is less variability in the objective function around the estimated minimum. The compression ratio at the observed minimum was 130:1.

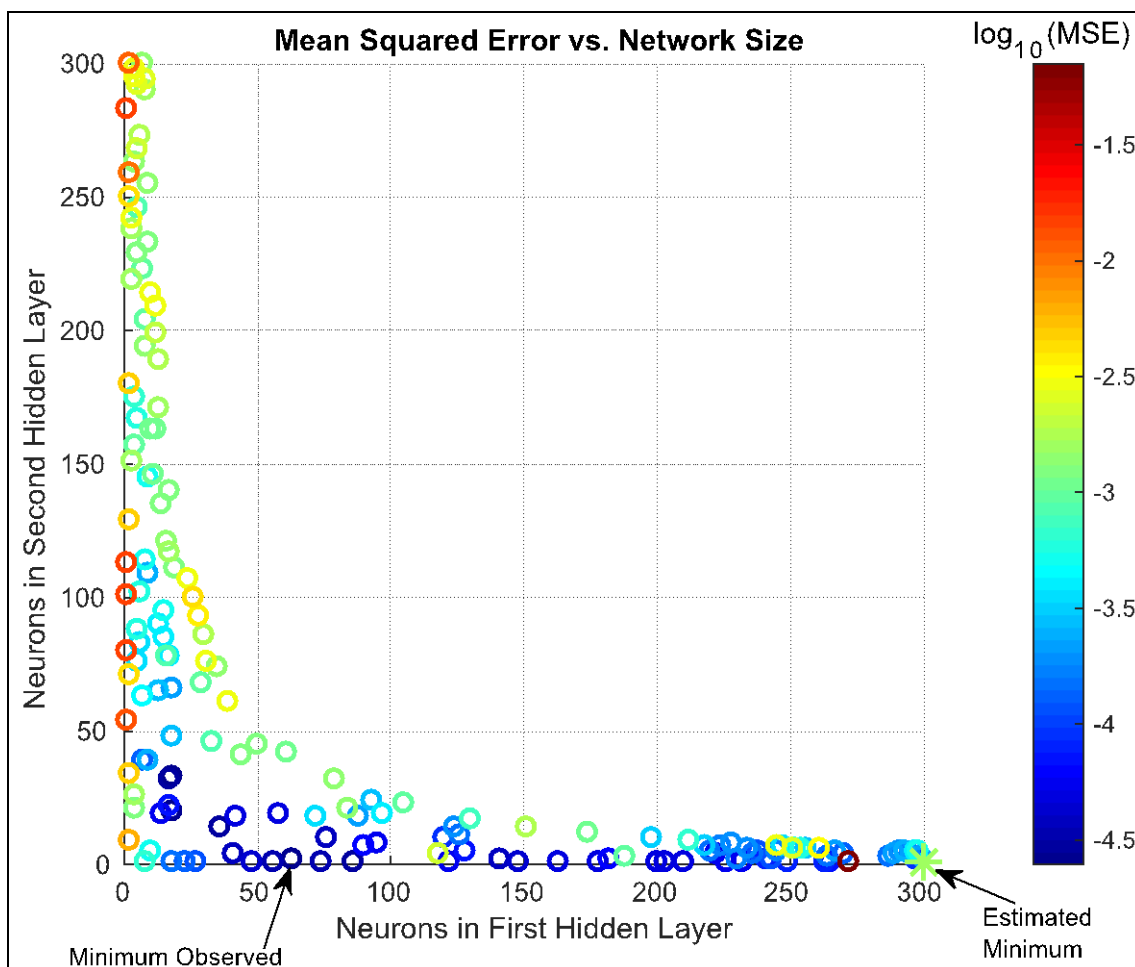


Figure 7: Effect of hidden layer sizes on training error.

One final Bayesian optimization was performed to better understand the tradeoffs between network size, accuracy, and training time. In this case:

- The objective function was  $\log_{10}(\text{MSE})$  on the training set of 66,339 P(I) values for the rear-fragmenting munition described in section 3.2.
- The hidden layer transfer function was hyperbolic tangent.
- The variable being optimized was the number of neurons in the first hidden layer. All networks had two neurons in their second hidden layer.
- The training method was Levenberg-Marquardt backpropagation with default parameters.
- Training times of 15, 30, 60, and 120 seconds were used.
- Larger networks were able to reduce the error further than smaller networks given more training time, but not by much. For instance, the most accurate network had 83 neurons in its first layer and was trained for 180 seconds, reaching an MSE of  $2.3 \times 10^{-5}$ , but another network with only 47 first-layer neurons achieved an MSE of  $5.2 \times 10^{-5}$  in just 15 seconds of training.

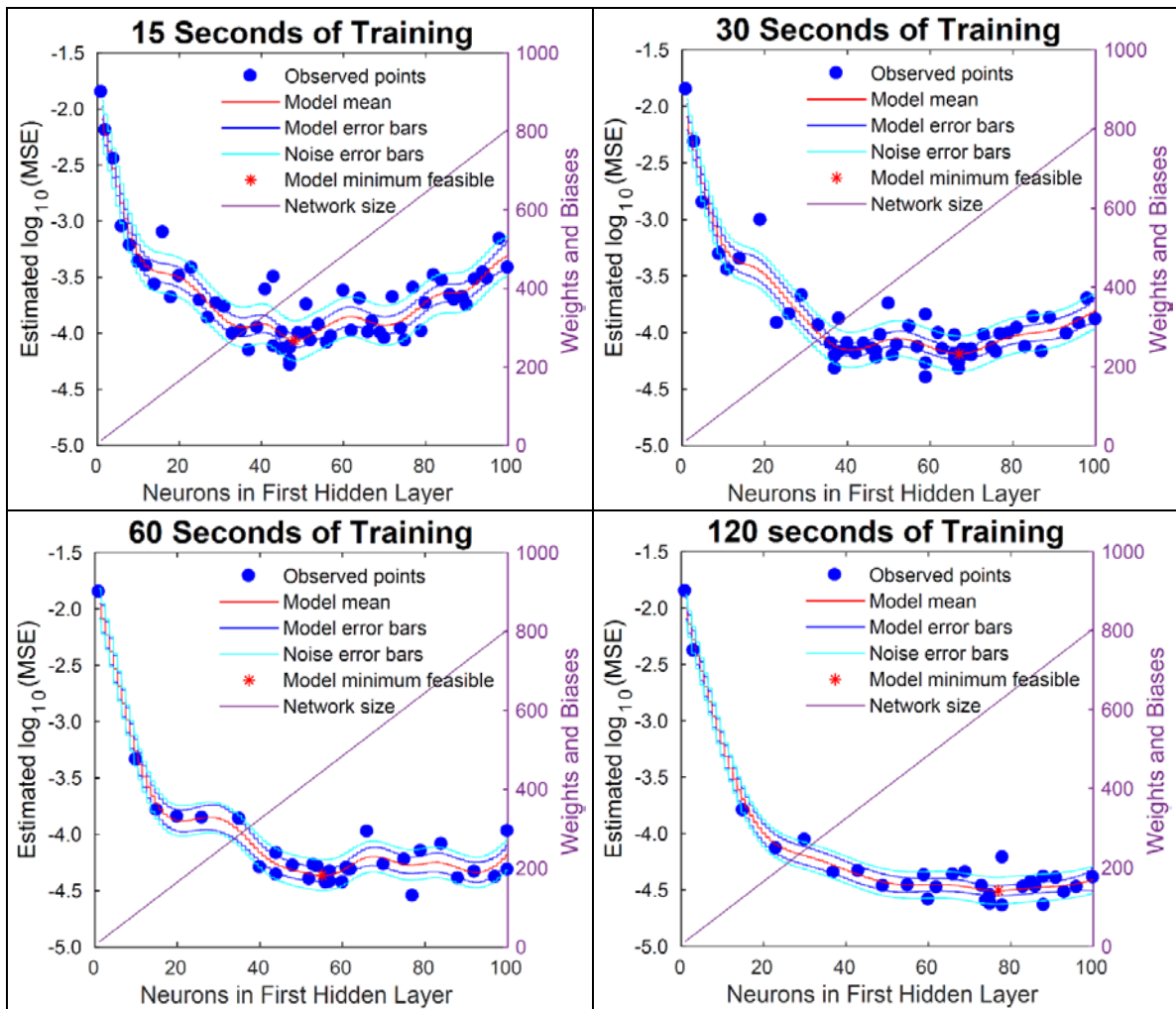


Figure 8: Bayesian optimization of network size for different training times.

## 2.4 Generalization and Data Augmentation

In the previous section, the metric used to assess the accuracy of a neural network was the mean squared error between the training data and the network’s approximation of the training data. One problem with this metric is that it does not assess how well a network is able to predict values that are not in the training data. The ability to predict novel values is known as generalization.

The 66,339 P(I) values used for training in the previous section were for 3 firer-to-target ranges, 9 burst heights, 27 burst-to-target ranges, and 91 azimuth angles. Intuitively, one might suspect that neural networks trained on this data would have a difficult time predicting P(I) values for novel firer-to-target ranges and burst heights, given the relative sparsity of those dimensions in the training data. Evidence supporting this suspicion is shown in Figure 9, which plots the “actual” P(I) values in the training data along with the P(I) values predicted by a neural network for a burst height of 0 meters, a burst-to-target range of 4.7018 meters, and an azimuth angle of 180°. The MSE between the actual and predicted values is  $7.4 \times 10^{-5}$ , but the MSE between the predicted values and the linear interpolation of the actual values is  $7.7 \times 10^{-4}$ , over 10 times larger.

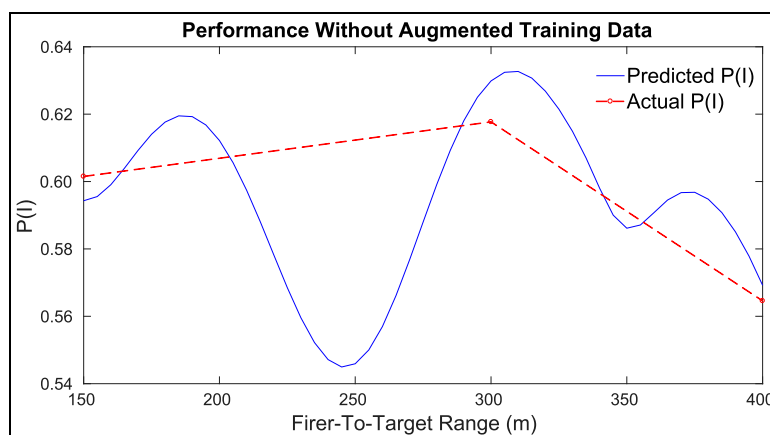


Figure 9: Neural network generalization without augmented training data.

To improve generalization, a new network was trained on linearly interpolated P(I) values for 18 firer-to-target ranges and 18 burst heights. This network was worse at predicting the three actual P(I) values shown in Figure 9, achieving an MSE of  $2.0 \times 10^{-4}$  instead of  $7.4 \times 10^{-5}$ , but it was better at predicting novel values, achieving an MSE relative to linear interpolation of  $2.5 \times 10^{-4}$  instead of  $7.7 \times 10^{-4}$ . The improved generalization of this new network is apparent from the plot shown in Figure 10.

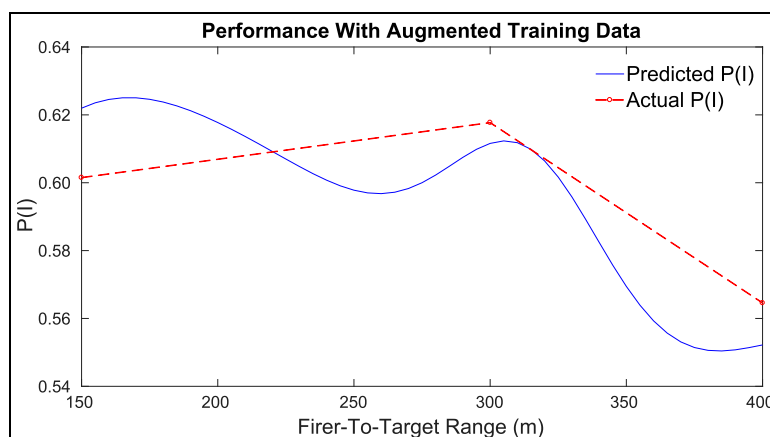


Figure 10: Neural network generalization with augmented training data.

Several other neural networks were trained on the same dataset augmented to varying degrees by linear interpolation. Otherwise, the training methods were identical, as were the network architectures:

- The original training data consisted of the 66,339 P(I) values for the rear-fragmenting munition described in section 3.2.
- The objective function was mean squared error.
- The hidden layer transfer function was hyperbolic tangent.
- There were twenty neurons in the first hidden layer and twenty neurons in the second hidden layer, resulting in a compression ratio of 123:1.
- Each network was trained for nine minutes using the Levenberg-Marquardt backpropagation method with Bayesian regularization and default parameters.

The MSE on the training data increased slightly as that data was augmented with additional interpolated values. However, as shown in Figure 11, the MSE decreased by a factor of 100 when measured relative to linearly interpolated P(I) values over one million randomly chosen combinations of firer-to-target range, burst height, burst-to-target range, and azimuth angle. The 95% confidence intervals were computed using the bootstrap bias corrected and percentile method with 2,000 resamplings. These intervals are for the first 100,000 points due to computational limitations, which means that they are larger than the true confidence intervals.

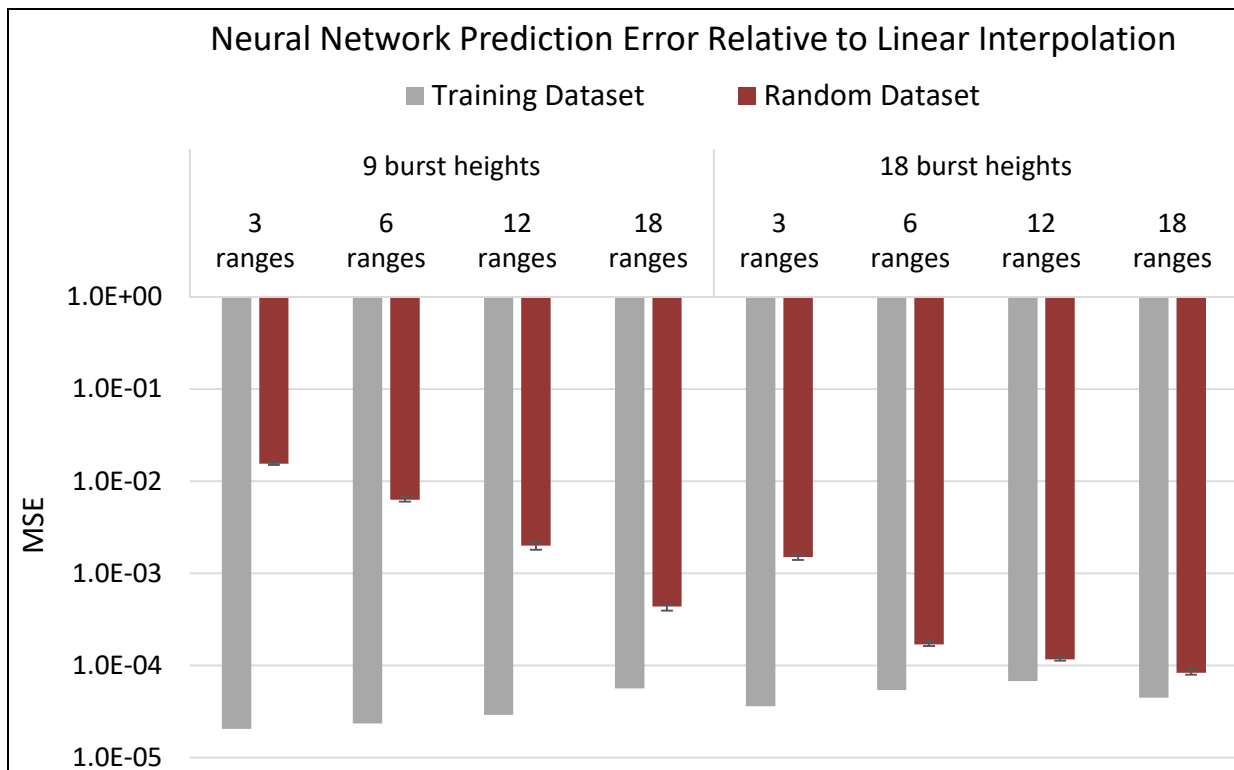


Figure 11: Improved generalization due to augmented training data.

### 3. EXAMPLE OPERATIONAL EFFECTIVENESS ANALYSIS

#### 3.1 Scenario Descriptions

##### 3.1.1 Exposed Target Scenario

In this scenario, the blue soldier fires one hypothetical, rear-fragmenting grenade at a point on the ground 5 meters behind the red soldier, as shown in Figure 12. The fuze is set to point detonate only, meaning that the grenade will only explode after hitting the ground. The red soldier is standing and the terrain is mostly flat. Variations of this scenario were created in which the firer-to-target range was 50, 100, 150, 200, 250, 300, 350, and 400 meters. The P(I) input format was also varied amongst four of the types defined in Table 1: whole-body grid, neural network, PIVR, and PIVR\_Angle. The direct-hit P(I) was set to one for all cases.



Figure 12: Overhead depiction of the exposed target scenario.

##### 3.1.2 Defilade Target Scenario

This scenario is similar to the exposed target scenario, except that in this case the target is completely hidden while standing 2 meters behind an impenetrable wall that is 20 meters wide, 0.5 meters thick, and 2.2 meters tall. The blue soldier is able to engage this target by turning on the hypothetical grenade’s programmable fuze and aiming 0.5 meters above the wall and 5 meters past the target, as shown in Figure 13.

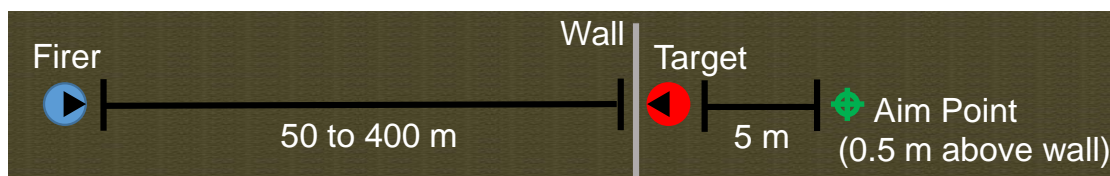


Figure 13: Overhead depiction of the defilade target scenario.

#### 3.2 Hypothetical Grenade

##### 3.2.1 Delivery Accuracy

The delivery errors for the hypothetical grenade were about 50% less than the typical delivery errors for a low velocity grenade. The fixed and variable biases were zero and the random errors are shown in Table 3.

Table 3: Hypothetical grenade delivery errors.

Range (m)	Random Error		
	Horizontal (mils)	Vertical (mils)	Range (m)
50	3.327	1.106	1.125
150	3.327	1.106	1.125
300	4.618	2.438	2.407
400	6.519	4.726	4.667

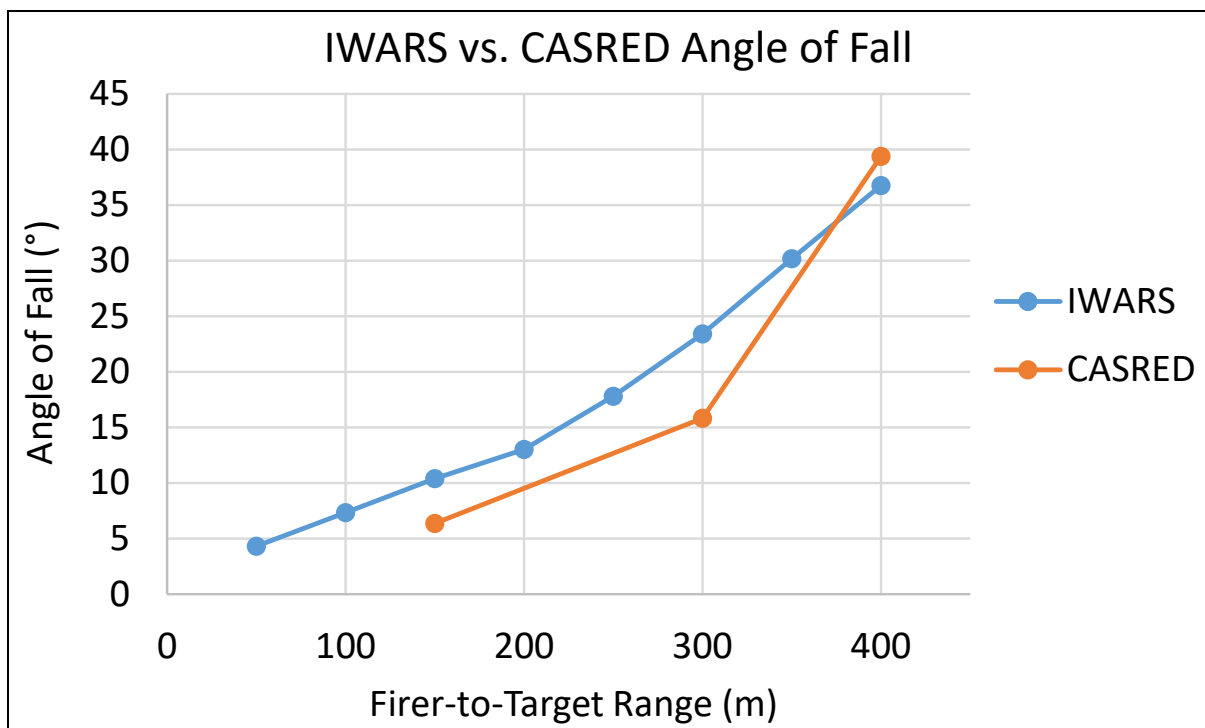
### 3.2.2 Terminal Conditions

The munition terminal velocity and angle of fall values used in CASRED are shown in Table 4. These values are typical for a low velocity grenade.

**Table 4: Munition terminal velocity and angle of fall used in CASRED.**

Range (m)	Velocity (m/s)	Angle of Fall (°)
150	78	6.36
300	68	15.81
400	60	39.38

The P(I) tables used by IWARS are indexed by angle of fall, not firer-to-target range. Therefore, the IWARS trajectory model parameters, namely the muzzle velocity and drag coefficient, were adjusted to approximately match the angles of fall used in CASRED. After some trial and error, a muzzle velocity of 77 m/s and a drag coefficient of 0.0014 were chosen, resulting in the angles of fall shown in Figure 14. Also note that the neural network angle of fall interpolation option was used to facilitate the comparison between neural network P(I) input and other types of P(I) input.



**Figure 14: Comparison of munition angle of fall in CASRED and IWARS.**

### 3.2.3 Fragmentation

CASRED requires munition fragmentation data as input, in “Zone data” (Z-data) format, which groups fragments into zones based upon the angle their paths make with the nose-to-tail axis of the munition. The entire 180° from the nose of the munition to its tail is split into angular zones, such as the zone between  $\theta_1$  and  $\theta_2$  shown in Figure 15.

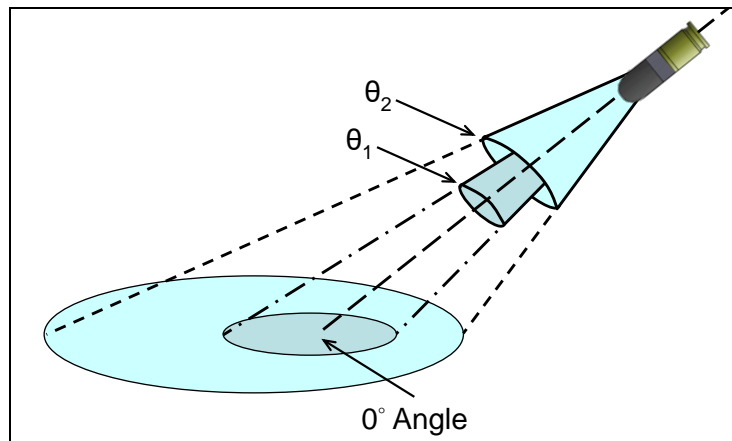


Figure 15: Angular zones of munition fragmentation data.

The fragmentation of the hypothetical, rear-fragmenting grenade is depicted in Figure 16. The height of the orange bar indicates that there are a total of 500 steel fragments in the 150° to 180° angular zone, and the orange color indicates that the fragments all weigh 4 grains. The fragment initial velocities of 2,000 feet/second are shown by the dashed line. The munition is represented in the center of the figure, with its nose pointing to the left as indicated by the arrow.

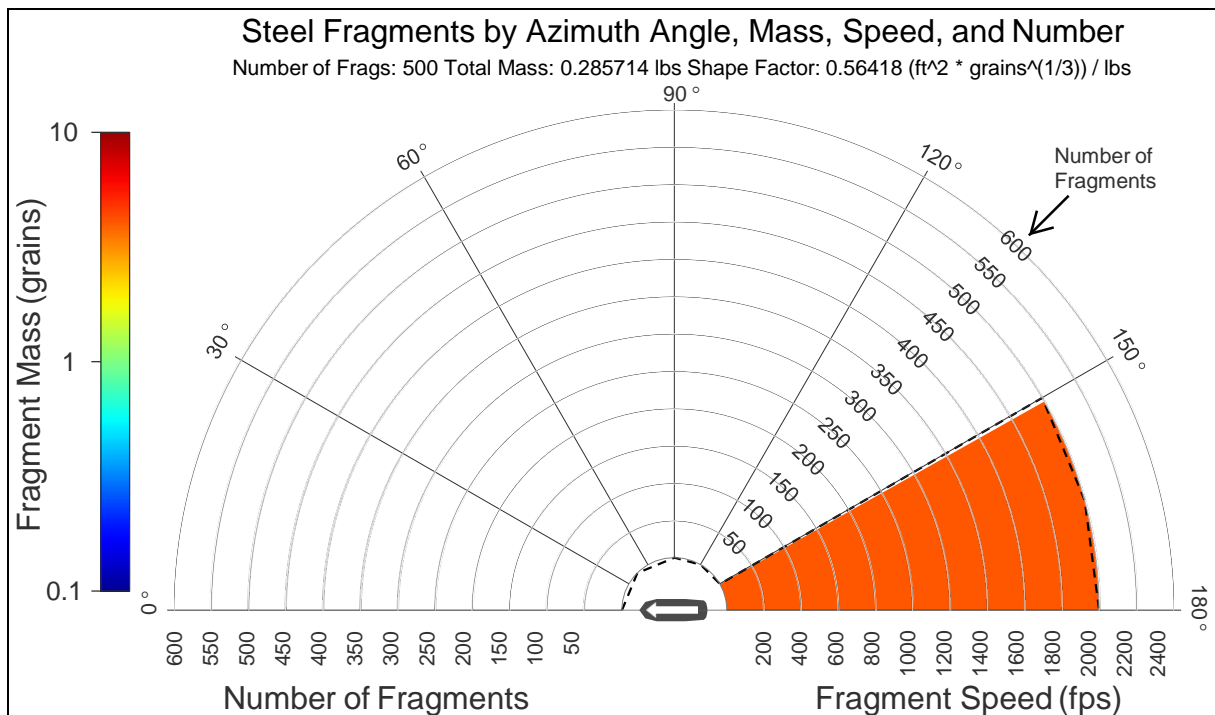


Figure 16: Fragmentation data used for the hypothetical grenade.



### 3.2.4 Terminal Effects

CASRED was used to estimate P(I) for the hypothetical grenade against a standing target not wearing any body armor. The casualty criterion was “30 second defense” and the output type was a whole-body grid with the parameters listed below.

- 3 firer-to-target ranges: 150, 300, and 400 meters.
- 9 burst heights: 0, 0.3048, 0.9144, 1.2192, 1.524, 1.8288, 2.7432, 4.572, and 6.096 meters (0, 1, 3, 4, 5, 6, 9, 15, and 20 feet).
- 27 burst-to-target ranges: 20 ranges logarithmically spaced from 0.3048 meters to 7.62 meters, and 7 linearly-spaced ranges from 7.62 meters to 14.344 meters.
- 91 azimuth angles: Every 2° from 0° to 180°. The P(I) values between 180° and 360° were assumed to mirror the P(I) values between 0° and 180°.

These P(I) values were compressed using the following techniques.

- Neural network: A neural network with 20 neurons in each of two hidden layers, trained on the original whole-body grid data augmented with interpolated P(I) values for firer-to-target ranges of 187.5, 225, 262.5, 325, 350, and 375 meters. The network was trained for 5 minutes using the Levenberg-Marquardt backpropagation method with Bayesian regularization and default parameters. The MSE performance of this network relative to linear interpolation on random data points was  $1.1 \times 10^{-3}$ .
- PIVR\_Angle: P(I) values averaged separately in each of three angular zones, from 0° to 144°, from 144° to 149°, and from 149° to 180°. As noted earlier, the P(I) values between 180° and 360° were assumed to mirror the P(I) values between 0° and 180°.
- PIVR: P(I) values averaged over all azimuth angles.

The following section will show the most relevant subset of P(I) input for each scenario, in each of the four formats (whole-body grid, neural network, PIVR\_Angle, and PIVR), followed by the IWARS results for that scenario.

## 3.3 Results

### 3.3.1 Exposed Target Scenario

The average aim height in the exposed target scenarios is about 0 meters. For munition bursts at this height, the neural network and PIVR\_Angle input types are both accurate representations of the original data, as shown in Figure 16. The accuracy of the PIVR\_Angle input type was not surprising because that type of input was created to represent munitions such as this one that have a simple, directional fragmentation pattern. In contrast, the PIVR input type is very inaccurate.

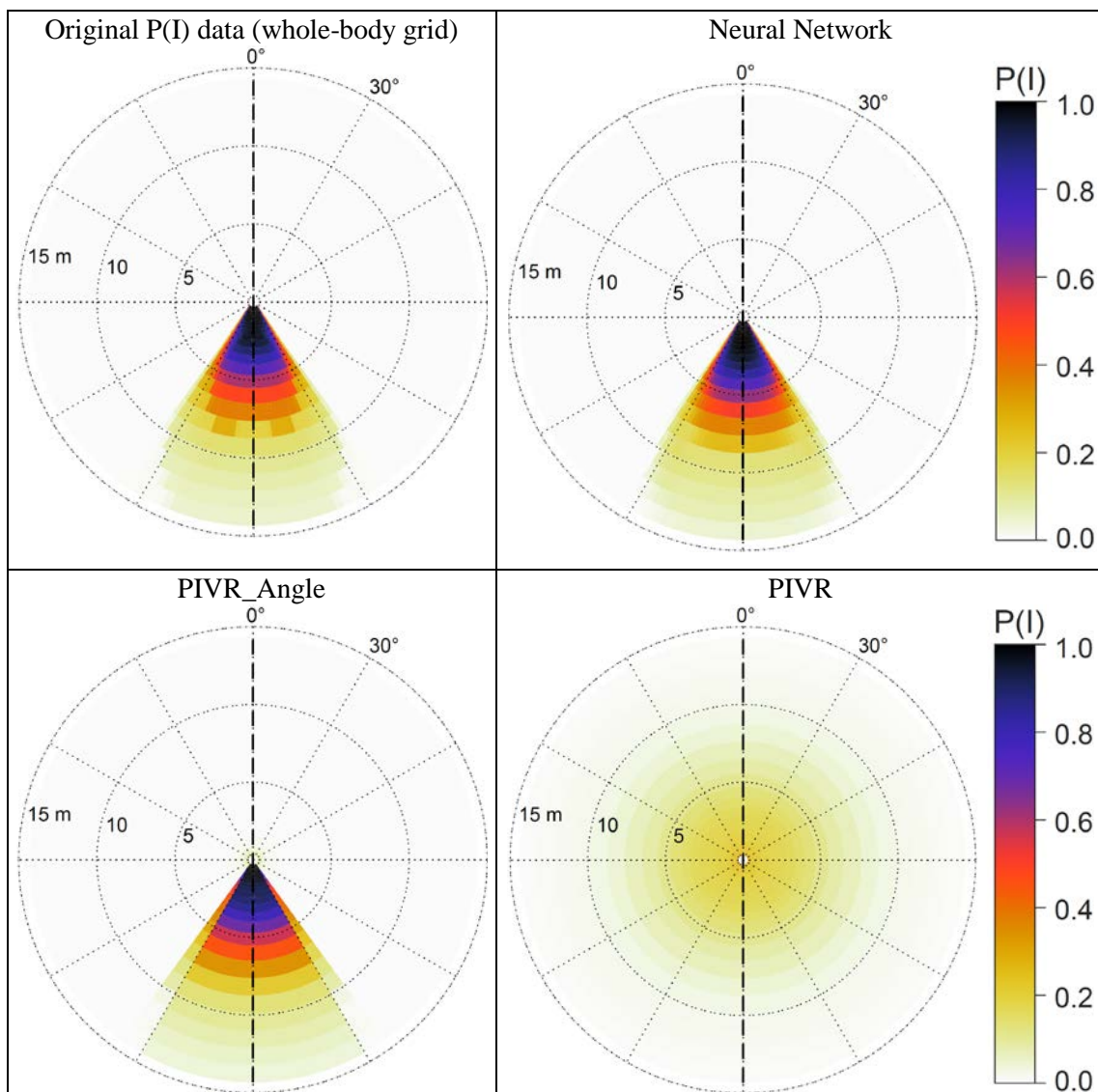


Figure 17: P(I) inputs to IWARS for 150 meters range and 0 meters burst height.

In these simple scenarios with only one shot and one target, expected fraction of casualties is equal to the probability of incapacitation given a shot, denoted  $P(I|Shot)$ . In terms of this metric, IWARS results for the exposed target scenarios were nearly identical for the whole-body grid (the original P(I) data), neural network, and PIVR\_Angle input options, whereas the PIVR input option was very inaccurate, as shown in Figure 18. These results are as predicted, given the visual comparisons of P(I) input shown in Figure 17. The slight dip in lethality at 200 meters range is due to a ridge behind the target that elevated the aim point by 1 meter. One thousand trials were run per case and the 95% confidence intervals were computed using the normal approximation method.

Figure 18 also shows the compression ratios for the different P(I) input types. Although the neural network and PIVR\_Angle input options were both very accurate, the neural network input is about 12 times smaller. Note that IWARS requires about twice as many P(I) values as were calculated by CASRED, since IWARS needs P(I) values from 180° to 360° as well as from 0° to 180°.

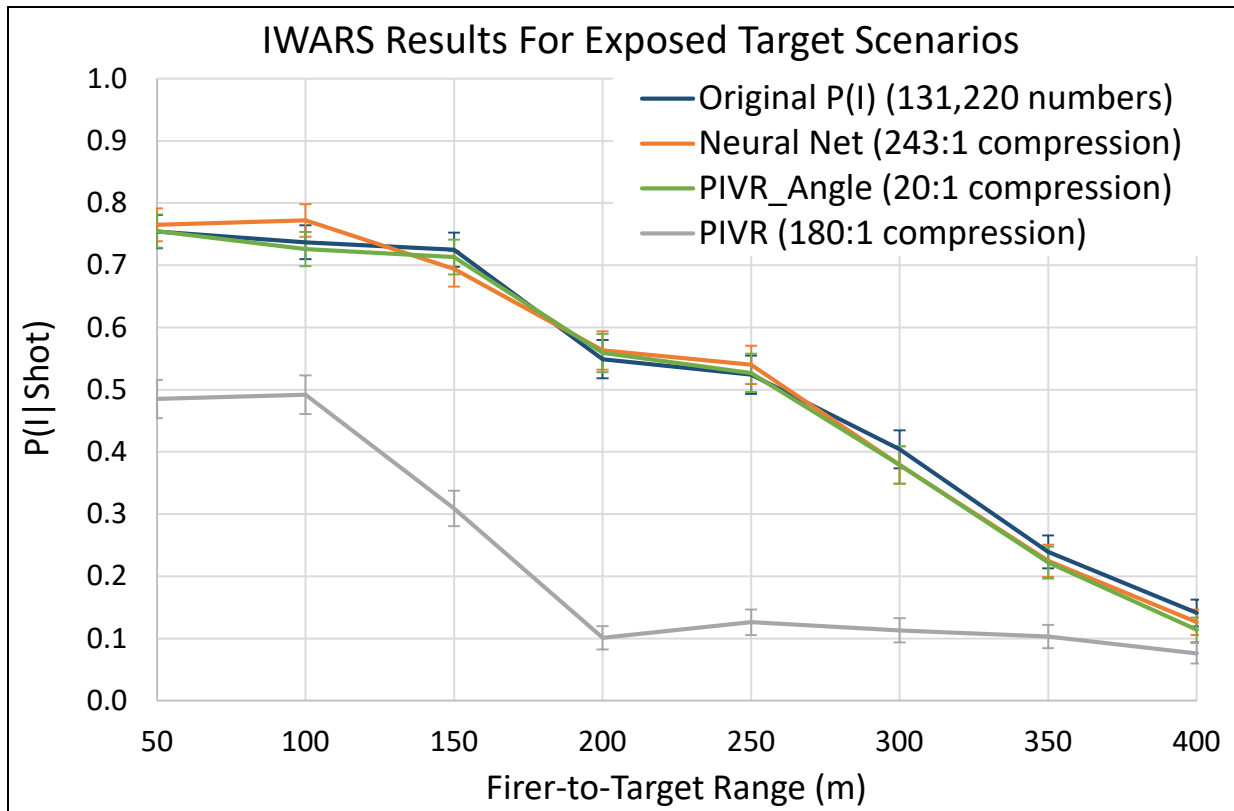


Figure 18: IWARS results for the exposed target scenarios.

### 3.3.2 Defilade Target Scenario

The average aim height in the defilade target scenarios is about 2.7 meters. For munition bursts at this height, the neural network input type is more accurate than the PIVR\_Angle input type with respect to representing the original data, as shown in Figure 19. The relatively poor accuracy of the PIVR\_Angle input type is due to the large variation in  $P(I)$  values across the  $150^\circ$  to  $180^\circ$  zone. The PIVR input type is by far the least accurate representation of the original data.

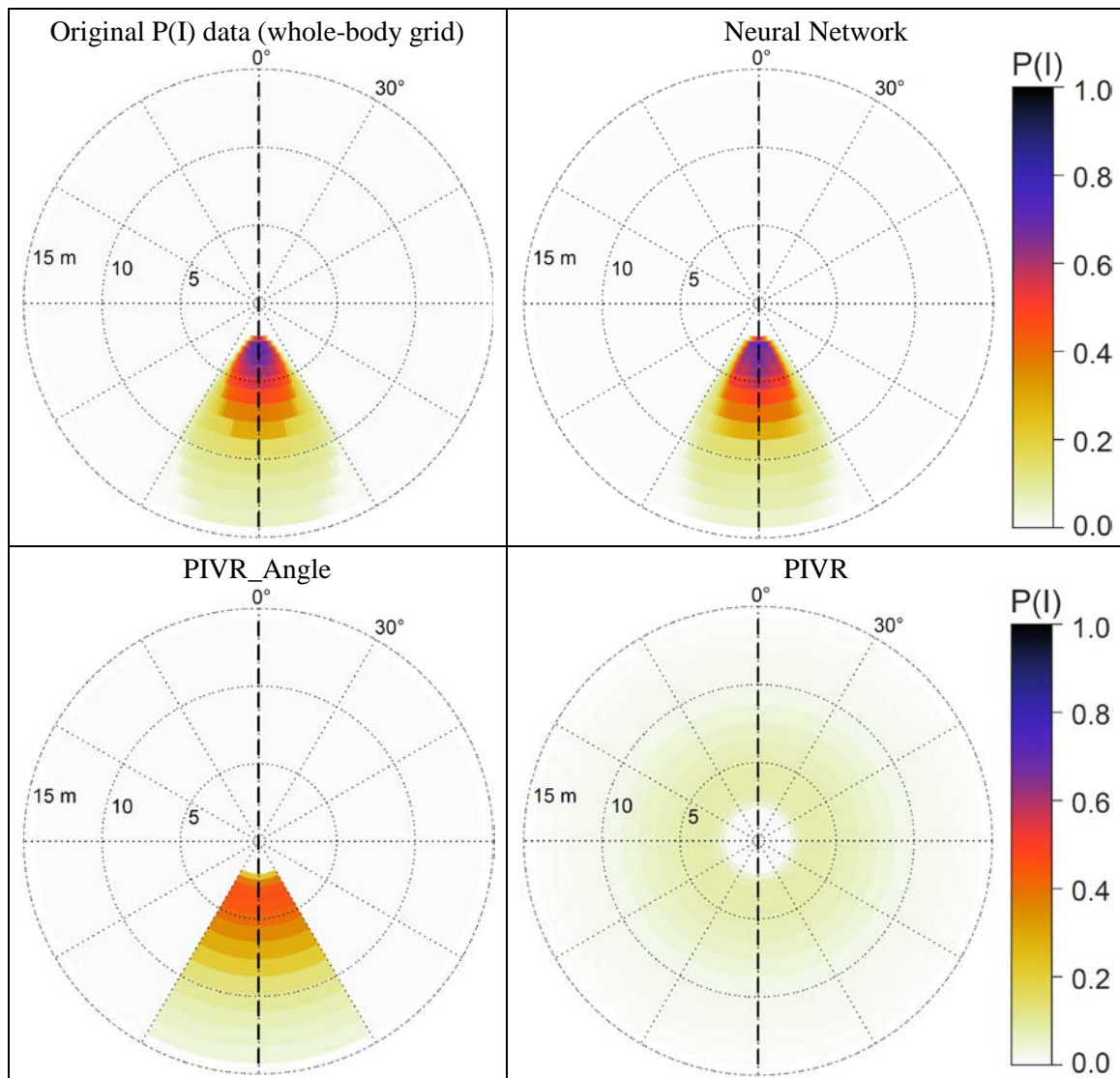


Figure 19:  $P(I)$  inputs to IWARS for 150 meters range and 2.7432 meters burst height.

IWARS results for the defilade target scenarios were nearly identical for the whole-body grid (the original P(I) data) and the neural network input options, as shown in Figure 20. The PIVR\_Angle input option was less accurate, particularly at shorter ranges, whereas the PIVR input option was very inaccurate at all ranges. These results are as predicted, given the visual comparisons of P(I) input shown in Figure 19.

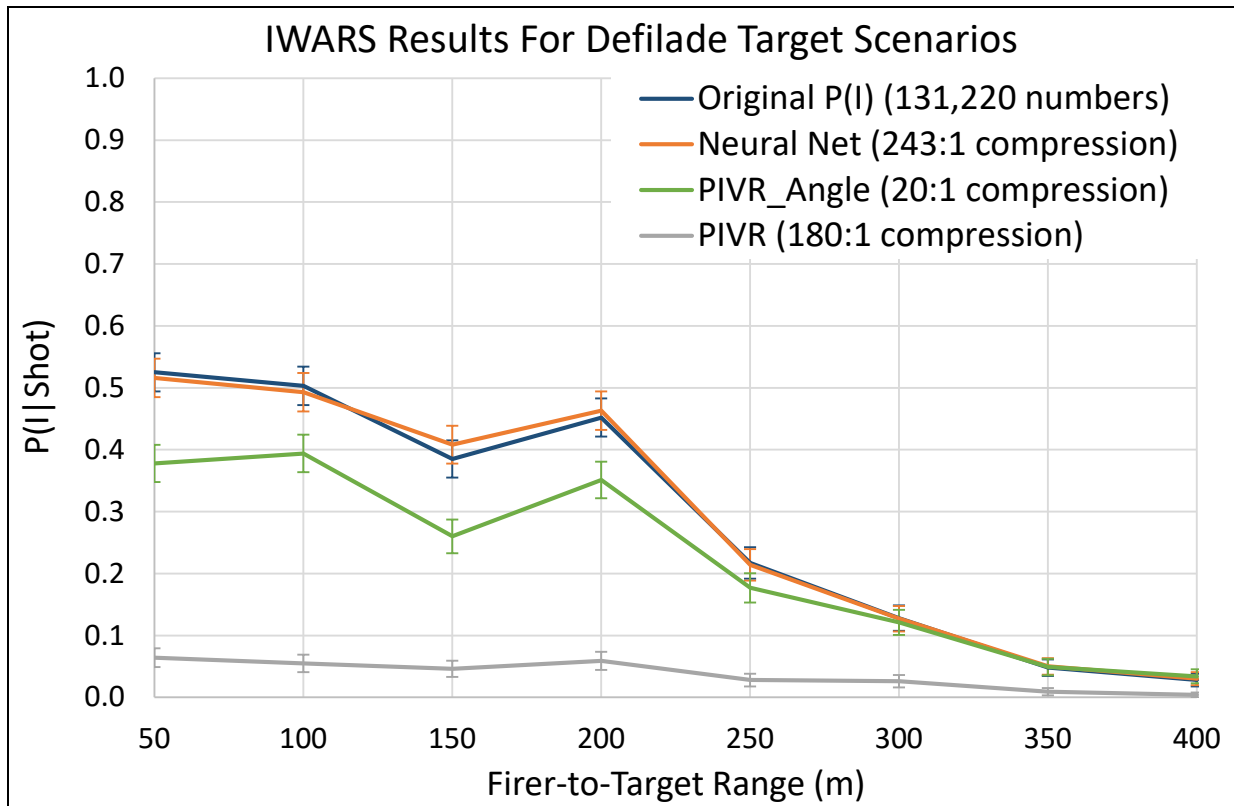


Figure 20: IWARS results for the defilade target scenarios.

### 3.3.3 IWARS Runtimes

Using neural network input was about as fast as using PIVR and PIVR\_Angle input, all of which were significantly faster than using the original P(I) data, as shown in Figure 21. These results are based on 100 trials per case and the 95% confidence intervals were computed using the normal approximation method.

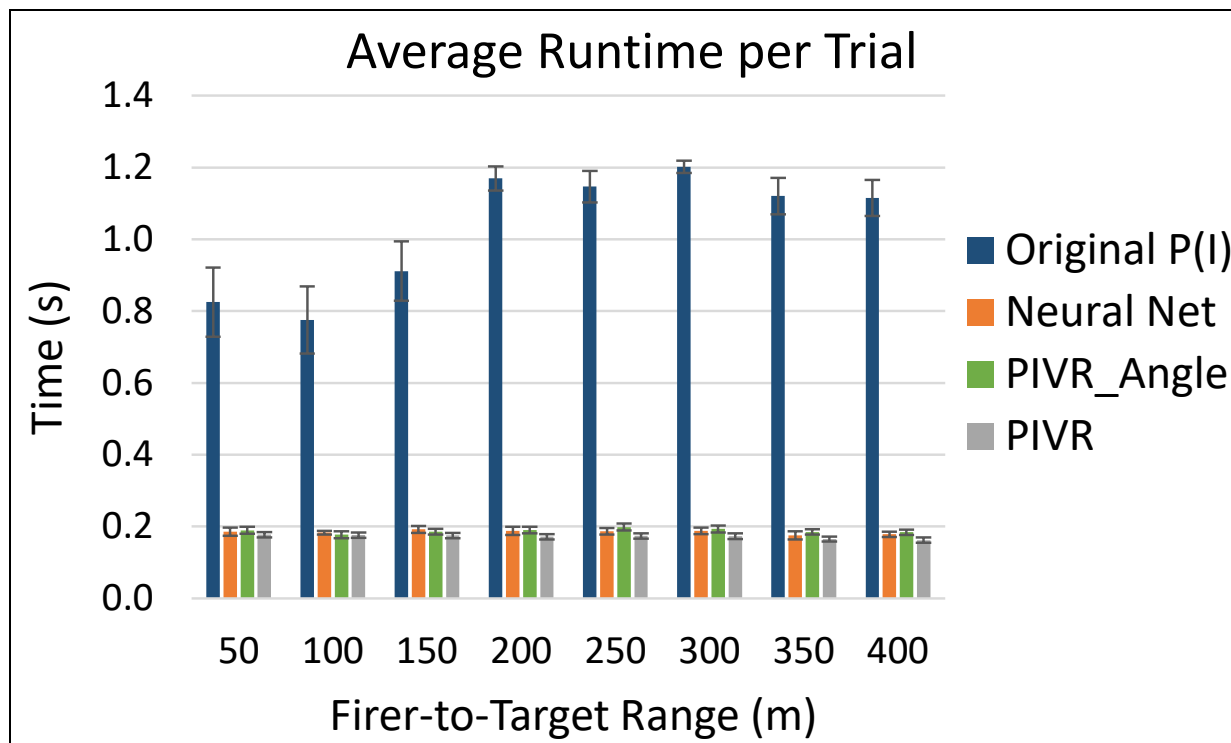


Figure 21: Average runtime per trial for the exposed target scenarios.

## 4. SUMMARY AND CONCLUSION

This paper describes a new method for compressing fragmenting munition lethality data using artificial neural networks. Using the neural network method, compression ratios of over 1000:1 were achieved with little loss of accuracy. It was shown that other compression techniques are either less accurate or have a lower compression ratio than the new method. The accuracy of the various compression techniques was evaluated by comparing the results of using both compressed and uncompressed lethality data in the simple, one-sided combat simulation known as FBAR.

Neural network compression worked well, even with manual tuning of model hyperparameters. However, the speed and accuracy of the compression was improved using techniques such as Bayesian optimization, data augmentation, and testing on random inputs.

Finally, a neural network P(I) input option was successfully integrated into IWARS, giving analysts the opportunity to use a highly compressed dataset without sacrificing accuracy or runtime. Previous input options are either less accurate or cannot be used in some circumstances due to a database size limitation.

## 5. REFERENCES

- [1] J. R. Way and J. Collins, “Advanced 40 mm Grenade Concepts,” U.S. Army Materiel Systems Analysis Activity, Aberdeen Proving Ground, MD, Jun 2018.
- [2] J. R. Way, “Operational Effectiveness Analysis of a Squad-Level Counter-Defilade Weapon,” U.S. Army Materiel Systems Analysis Activity, Aberdeen Proving Ground, MD, Jan 2019.
- [3] J. R. Way, “Lethality Analysis of Several Air-Bursting, Low-Velocity 40 mm Grenades Against Personnel Targets,” U.S. Army Combat Capabilities Development Command – Data and Analysis Center, Aberdeen Proving Ground, MD, Pending Publication.
- [4] J. R. Way, “Fragmenting Munition Lethality Data Compression By Neural Network Regression,” U.S. Army Materiel Systems Analysis Activity, Aberdeen Proving Ground, MD, Nov 2015.
- [5] J. R. Way, “Using Neural Networks to Compress Grenade Lethality Data in the Infantry Warrior Simulation (IWARS),” DEVCOM Data & Analysis Center, Aberdeen Proving Ground, MD, Sep 2019.
- [6] M. H. Beale, M. T. Hagan and H. B. Demuth, “Neural Network Toolbox User's Guide,” MathWorks, Natick, MA, Oct 2014.
- [7] M. A. Rosenblatt, “FBAR Version 5.5 User's/Analyst's Manual,” Army Materiel Systems Analysis Activity, Aberdeen Proving Ground, MD, May 2009.
- [8] P. I. Frazier, “A Tutorial on Bayesian Optimization,” 8 Jul 2018. [Online]. Available: arXiv:1807.02811 [stat.ML].
- [9] M. T. Hagan and M. Menhaj, “Training feed-forward networks with the Marquardt algorithm,” *IEEE Transactions on Neural Networks*, vol. 5, no. 6, pp. 989-993, 1999.
- [10] F. D. Foresee and M. T. Hagan, “Gauss-Newton approximation to Bayesian learning,” in *Proceedings of the International Joint Conference on Neural Networks*, Jun 1997.
- [11] J. E. Dennis and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Englewood Cliffs, NJ: Prentice-Hall, 1983.
- [12] Wikipedia, “Nonlinear conjugate gradient method,” [Online]. Available: [https://en.wikipedia.org/wiki/Nonlinear\\_conjugate\\_gradient\\_method](https://en.wikipedia.org/wiki/Nonlinear_conjugate_gradient_method). [Accessed 11 April 2019].
- [13] M. R. Hestenes and E. Stiefel, “Methods of Conjugate Gradients for Solving Linear Systems,” *Journal of Research of the National Bureau of Standards*, vol. 49, no. 6, pp. 409-436, Dec 1952.
- [14] M. J. D. Powell, “Restart procedures for the conjugate gradient method,” *Mathematical Programming*, vol. 12, pp. 241-254, 1977.

- [15] R. Fletcher and C. M. Reeves, "Function minimization by conjugate gradients," *Computer Journal*, vol. 7, pp. 149-154, 1964.
- [16] E. Polak and G. Ribière, "Note sur la convergence de méthodes de directions conjuguées," *Rev. Francaise Informat Recherche Operationelle*, vol. 3, no. R1, pp. 35-43, 1969.
- [17] M. F. Moller, "A Scaled Conjugate Gradient Algorithm For Fast Supervised Learning," *Neural Networks*, vol. 6, no. 4, pp. 525-533, Dec 1993.
- [18] M. Riedmiller and H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm," in *IEEE International Conference on Neural Networks*, San Francisco, CA, Mar 1993.
- [19] MathWorks, "traingdx," MathWorks, [Online].  
Available: <https://www.mathworks.com/help/deeplearning/ref/traingdx.html>. [Accessed 17 April 2019].