Simultaneous Localization of Multiple GNSS Interference Sources via Neural Networks

David Besson, *GPS Directorate*

BIOGRAPHY

David Besson is a Captain in the United States Air Force and serves as the domestic spectrum management engineer for the Global Positioning Systems Directorate located at the Space and Missile Systems Center, Los Angeles Air Force Base, California. He holds a BS in Astronautical Engineering from the United States Air Force Academy and an MS in Aeronautics and Astronautics from the University of Washington. His work in graduate school focused on space-based applications of modern control and estimation theory. The views expressed in this paper are those of the author and do not reflect the official policy or position of the United States Air Force, United States Department of Defense, or United States Government.

ABSTRACT

The accuracy and availability of Global Navigation Satellite System (GNSS) applications hinge on the rapid localization and mitigation of interference sources. This research proposes to localize multiple interference sources simultaneously using artificial neural networks; one of the primary tools in the rapidly expanding field of machine learning. In particular, we leverage the techniques of handwritten character recognition to reconstruct the localization problem in a tractable multi-label, multi-class classification framework. We pose the multiple GNSS interference source localization problem as determining which of 400 cells in a 100 km^2 grid contain transmitters interfering with the Global Positioning System (GPS) L1 signal based on in-band power measurements taken by 25 sensors from an array of known, fixed positions. In this preliminary work we only consider stationary and isotropic interference sources and train the neural network to simultaneously classify the cell positions of up to ten 100 W interference sources. We develop the training sets for the neural network by simulating the aggregate interference of tens of thousands of interference source combinations. The aggregate interference is recorded by simulated drones that are hovering at a fixed altitude of 121.92 m (400 ft) above the search grid. These measurements roughly correspond to the array of pixel intensities that are fed into neural networks for handwritten character recognition. The neural network in our problem is comprised of five layers: the input layer of 25 aggregate interference power measurements, three hidden layers with 100 perceptrons each, and the output layer of 400 values corresponding to the probabilities that each of the 400 cells contain interference sources. The output layer identifies only the probability that the cells contain interference sources, not the location of the sources within the cells or the transmitted interference power. The precision score of the trained neural network is approximately 80%. We simulate this approach using Python 2.7 and its scikit-learn 0.18 machine learning library, both of which are free software packages.

INTRODUCTION

The accuracy and availability of Global Navigation Satellite System (GNSS) applications hinge on the rapid localization and mitigation of interference sources. In this paper we propose to localize multiple interference sources simultaneously using neural networks; one of the primary tools from the rapidly expanding field of machine learning. In particular, we leverage the techniques of handwritten character recognition to reconstruct the localization problem in a tractable multi-label, multi-class classification framework.

This paper gives an overview of the fundamentals of neural networks, describes a process for creating simulated training and test sets for this specific problem formulation, presents the performance of the neural network relative to the training and test sets, and proposes conclusions and possible future work based on this research.

The problem formulation is as follows and illustrated in Figure 1. Suppose that a bounded two-dimensional region is broken into a grid of cells. Some of these cells contain interference sources. Above the search region is a set of fixed observation points. Radio-frequency measurements are collected at these observation points (either by drones, helicopters or by some other means) and relayed back to a central computer. We describe our approach as simultaneous localization because we are not attempting to localize the interference sources sequentially. Instead we propose to process the 25 raw sensor measurements as a single input vector in order to localize multiple interference sources at the same time by passing the input vector though the trained neural network. In other words, the sensor platforms are not doing any localization calculations. The calculations are performed on a separate, central computer that receives the raw sensor measurements, feeds them into a trained neural network, and outputs the predicted cells that contain the interference sources. The neural network does not know a priori how many interference sources to expect but the accuracy of the network's predictions largely depend on the number of simultaneous interference source combinations that it processed during training.

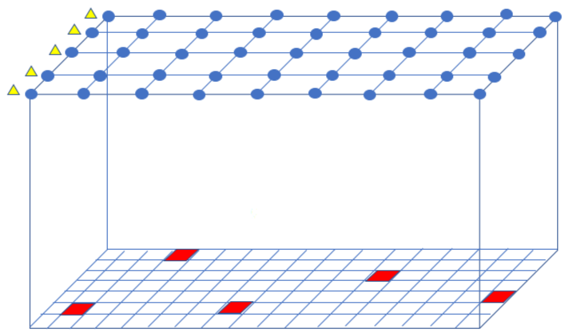


Figure 1: A simple illustration of the multiple interference source localization problem. The red tiles represent cells of the grid that contain interference sources, the yellow triangles represent drones equipped with interference measurement sensors, and the blue dots represent the points at which the drones will collect sensor measurements.

RELATED WORK

Hussein et al. wrote a helpful paper that summarizes the majority of the techniques and algorithms that have been applied to the interference localization problem for wireless sensor networks [1]. While these techniques were not designed with GNSS interference in mind, they are still helpful for determining how a neural network based approach fits into the state of the art for the interference source localization problem. The vast majority of the techniques covered in the survey - namely wireless network topology based localization, bounding circle localization, jammed nodes based localization, and jammer power range based localization - focus on locating a single interference source. Only two of the 13 surveyed techniques are intended to locate multiple sources.

The first is work done by Liu et al. in which they "devise an estimation scheme based on ambient noise floor and validate it with real-world experiments. To further reduce estimation errors, [they] define an evaluation feedback metric to quantify the estimation errors and formulate jammer localization as a nonlinear optimization problem, whose global optimal solution is close to jammers' true positions. [They] explore several heuristic search algorithms for approaching the global optimal solution" [2].

And the second is work done by Cheng et al. in which they "solve a multi-jammer localization problem, where multiple jammers launch collaborative jamming attacks. [They] develop an x-rayed jammed-area localization (X-ray) algorithm which skeletonizes jammed areas and estimates the jammer locations based on bifurcation points on skeletons of jammed areas. [Their] extensive simulation results demonstrate that with one run of the algorithms, X-ray is efficient in localizing multiple jammers in WSN with small errors" [3].

These two approaches rely on a combination of analytical estimation and heuristic techniques which sets them apart from the empirically (though at this stage only through simulation) based neural network approach we propose in this paper. We also treat the search space of interference source locations as a discrete grid while the two approaches described above generate location predictions in a continuous search space.

A separate body of work focuses on GNSS interference source localization specifically. Some notable examples are the Generalized Interference Detection and Localization System proposed by Gromov et al. [4], the network of low cost front-end modules for GNSS interference detection proposed by Lindström et al. [5], and adaptive antenna arrays for GPS interference localization as proposed by Trinkle et al [6]. These techniques exploit the unique characteristics of the GNSS signals and future work with neural network based localization should certainly look for ways to leverage connections with this existing - and growing - body of work. This paper does not address any of the unique properties of GNSS signals in order to focus the scope of the proposed work on the structure of a novel neural network based problem formulation.

The only aspect of the GNSS environment that this paper exploits is that the power levels of the GNSS signals are extremely low, so we can safely assume that the received power of the interfering signals is significantly greater than that of the true GNSS signals. The problem would be more difficult in a noisier spectral environment in which the true signals are closer in power to the interfering signals.

BRIEF OVERVIEW OF ARTIFICIAL NEURAL NETWORKS

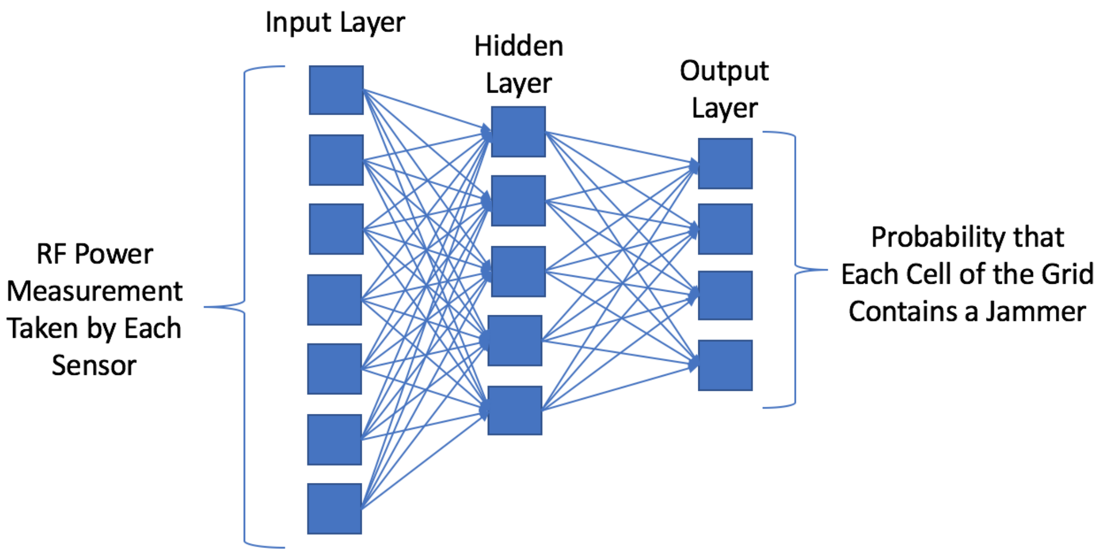

Input Layer
Hidden Layer
Output Layer
RF Power Measurement Taken by Each Sensor
Probability that Each Cell of the Grid Contains a Jammer

Figure 2: A simple illustration of an artificial neural network. Our actual neural network contains 25 nodes in the input layer, 100 nodes in each of its three hidden layers, and 400 nodes in the output layer.

We now explain the mathematical fundamentals of the artificial neural network. An excellent overview of artificial neural networks is given in [7] and we briefly summarize it in the remainder of this section. A more practical online crash course on artificial neural networks and other machine learning techniques is given in [8].

A mathematical neuron, otherwise known as a perceptron, in an artificial neural network computes a weighted sum of its n input signals, x_j where $j = 1, 2, ..., n$, feeds this sum into an activation function, and generates an output if this sum is above a certain threshold u. This output then feeds feeds into all of the nodes in the next layer of the network. Mathematically,

$$y = g\left(\sum_{j=1}^{n} w_j x_j - u\right), \tag{1}$$

where $g(\cdot)$ is an activation function and w_j is the neuron weight associated with the jth input. We use the rectifier activation function which is given as

$$g(x) = \max(0, x). \tag{2}$$

Now let $\{(\mathbf{x}^{(1)}, \mathbf{d}^{(1)}), (\mathbf{x}^{(2)}, \mathbf{d}^{(2)}), ..., (\mathbf{x}^{(p)}, \mathbf{d}^{(p)})\}$ be a set of p training patterns (input-output pairs), where $\mathbf{x}^{(i)} \in R^n$ is the input vector in the n-dimensional pattern space, and $\mathbf{d}^{(i)} \in [0,1]^m$, an m-dimensional hypercube. For classification purposes, m is the number of classes. Our neural network is multi-class because $m > 2$ and it is multi-label because inputs can be assigned to more than one class. Physically this means that we permit the input vectors of raw sensor measurements to map to either a single interference source or multiple interference sources.

The squared-error cost function most commonly used in the artificial neural network literature is defined as

$$E = \frac{1}{2} \sum_{i=1}^{p} \left\| \mathbf{y}^{(i)} - \mathbf{d}^{(i)} \right\|^2. \tag{3}$$

The weights w_j are tuned using a back-propagation algorithm. This is a gradient-decent method to minimize the squared error cost function in Equation 3. For our simulations we use the Adam stochastic optimization method proposed in [9]. We summarize the steps in a general back-propagation algorithm below:

1. Initialize the weights to small random values.
2. Randomly choose an input patter $\mathbf{x}^{(\mu)}$.
3. Propagate the signal forward through the network.
4. Compute δ_i^L in the output layer
$$\delta_i^L = g'(h_i^L)[d_i^u - y_i^L], \tag{4}$$
where h_i^L represents the net input to the ith unit in the lth layer, L is the number of layers in the neural network, and g' is the derivative of the activation function g.
5. Compute the deltas for the preceding layers by propagating the errors backwards;
$$\delta_i^l = g'(h_i^l) \sum_j w_{ij}^{l+1} \delta_j^{l+1}, \tag{5}$$
for $l = (L-1), ..., 1$.
6. Update weights using
$$\Delta w_{ji}^l = \eta \delta_i^l y_j^{l-1} \tag{6}$$
where η is the learning rate.
7. Go to step 2 and repeat for the next pattern until the error in the output layer is below a prespecified threshold or a maximum number of iterations is reached.

The artificial neural network is ready to make predictions on new samples once the weights w_j have been tuned.

We show a simplified version of our artificial neural network in Figure 2. Note that our actual neural network contains 25 nodes in the input layer, 100 nodes in each of its three hidden layers, and 400 nodes in the output layer. We have not optimized the macro parameters of the neural network such as the number of hidden layers, number of nodes in each hidden layer, and the learning rate η. Our goal was to create a functional neural network and leave the optimization of the network's performance to future research.

CREATION OF TRAINING SETS AND TEST SETS

The foundation of the simulated training sets is the simplified form of the Friis transmission equation which is given as

$$P_r = P_t G_t G_r \left(\frac{\lambda}{4\pi R} \right)^2 \tag{7}$$

where P_r is the is the input power at the receiver antenna, P_t is the output power of the transmitting antenna, G_t and G_r are the antenna gains (with respect to an isotropic radiator) of the transmitting and receiving antennas respectively, λ is the wavelength, and R is the distance between the antennas. The following values are used for all simulations in this research

1. $P_t = [50, 150]$ W
2. $G_r = G_t = 1$
3. $\lambda = \frac{c}{f_{\text{L1}}} = 0.190293672798$ m

where c is the speed of light (299792458 m/s) and f_{L1} is the frequency of the GPS L1 signal (1575.42 MHz). The first step in the training set creation process is to choose a search area. For these simulations, the search area is the region surrounding Portland, Oregon. This area is chosen to be $10,000$ m $\times$ $10,000$ m and is shown, to scale, in Figure 3 below.

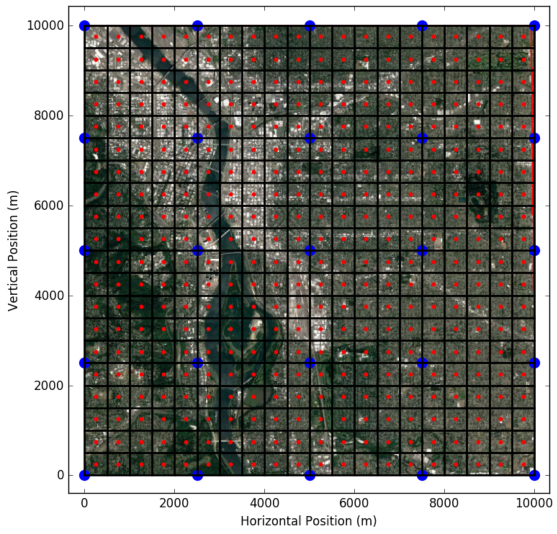


Figure 3: A 100 km^2 region surrounding Portland, Oregon. The black grid represents the cells in the search grid, the red dots represent training jammers, and the blue dots represent aggregate interference measurement collection points. We use an image taken from Google Earth.

Next, we divide the region into a grid of 400 equally-sized square cells so that each cell has an area of 0.25 km^2. Now we create a set of training interference sources by placing an interference source in the center of each cell. We also place 25 aggregate interference measurement collection points throughout the search region at an altitude of 121.92 m (400 ft). We illustrate the resulting cells (black grid), training jammers (red dots), and sensors (blue dots) in Figure 3.

Now the simulation must calculate the distances between each of the training interference sources and each of the sensors and then the power of each interference source that is received by each sensor (one at a time). In this example, there are 400 training interference sources and 25 sensors, so this corresponds to $10,000$ calculations of R and $10,000$ calculations of P_r. The equation for the distance $R_{i,k}$ between sensor i and interference source k is simply

$$R_{i,k} = \sqrt{(x_{s_i} - x_{j_k})^2 + (y_{s_i} - y_{j_k})^2 + (z_{s_i} - z_{j_k})^2} \tag{8}$$

where $(x_{s_i}, y_{s_i}, z_{s_i})$ are the coordinates of sensor i and $(x_{j_k}, y_{j_k}, z_{j_k})$ are the coordinates of interference source k. Each $R_{i,k}$ is then plugged into Equation 7 to get the power received by sensor i from interference source k, $P_{r_{i,k}}$.

Now we create combinations of the interference sources. We show the total number of possible interference source combinations in Table 1, where the equation for ${}_xC_y$ is the combination equation given by:

$${}_xC_y = \frac{x!}{y!(x-y)!}. \tag{9}$$

Table 1: Total number of possible interference source combinations.

${}_{400}C_1 = 400$	${}_{400}C_6 = 5.479 \times 10^{12}$
${}_{400}C_2 = 79,800$	${}_{400}C_7 = 3.084 \times 10^{14}$
${}_{400}C_3 = 10,586,800$	${}_{400}C_8 = 1.515 \times 10^{16}$
${}_{400}C_4 = 1,050,739,900$	${}_{400}C_9 = 6.598 \times 10^{17}$
${}_{400}C_5 = 83,218,600,080$	${}_{400}C_{10} = 2.580 \times 10^{19}$

Clearly it quickly becomes impractical to create and analyze every possible combination of interference sources. Fortunately, the neural network still performs well even if it is only trained with a subset of each total combination set. We address the impact of the training set size in in the next section.

Now, for each combination of interference sources, the simulator calculates the aggregate received power from all of the interference sources at each measurement point. The final step in the training set generation process is to create the inputs and outputs for the neural network. The inputs correspond to an array of the sensor measurements for each of the training combinations. The outputs correspond to the binary vector of cells containing interference sources for each of the training combinations. So in this case $\mathbf{x}_i \in R^{25}$ and $\mathbf{y}_i \in [0,1]^{400}$. The number of training samples is determined by the chosen number of training combinations, n_c. Therefore the final inputs and outputs are $\mathbf{X} \in R^{n_c \times 25}$ and $\mathbf{Y} \in [0,1]^{n_c \times 400}$.

CREATION OF TRAINING AND TEST SETS

The neural networks used in the following simulations are all comprised of three hidden layers that each contain 100 neurons. The positions of the interference sources in the training set are all in the exact center of the [500m × 500m] grid cells. There are 25 sensors distributed uniformly across the [10000m × 10000m] search area. The sensors are at an altitude of 121.92 m (400 ft) which is the current legal altitude limit for drones as detailed in [10]. We train the network with a training set comprised of groups of $[1, \ldots, 10]$ interference sources. Each of the interference sources is transmitting at 100 W. We vary the performance of the neural network by gradually increasing the size of the training sets. We construct the training sets in the following manner:

1. All training sets contain the initial ${}_{400}C_1$ list of single interference sources.

2. For the remaining combination sets ${}_{400}C_n | n = [2, \ldots, 10]$, we choose a random subset of a fixed size m, where $m = [10,\ 30,\ 100\ 300,\ 1000,\ 3000,\ 10000,\ 30000]$.

We show the total sizes of the training sets in Table 2.

Table 2: Size of training sets.

m	Total Size of Training Set (n_c)	m	Total Size of Training Set (n_c)
10	490	1000	9400
30	670	3000	27400
100	1300	10000	90400
300	3100	30000	270400

We also create two additional test sets. The first is based on the same 400 single interference sources used in the training sets but we choose different random subsets of each of the larger combination sets. Later in the paper we refer to this as the easy test set. The other set is meant to be much more difficult for the neural network and is illustrated in Figure 4. In this set we vary the position of the interference sources within their grid cells, vary the altitude of the interference sources by ±3 m, vary the transmitter power of each interference source by ±50W, and the sensors experience up to 2 m of positioning error. Later in the paper we refer to this as the hard test set. Both of these test sets are set to $m = 1000$ so that they each contain 9400 test samples.

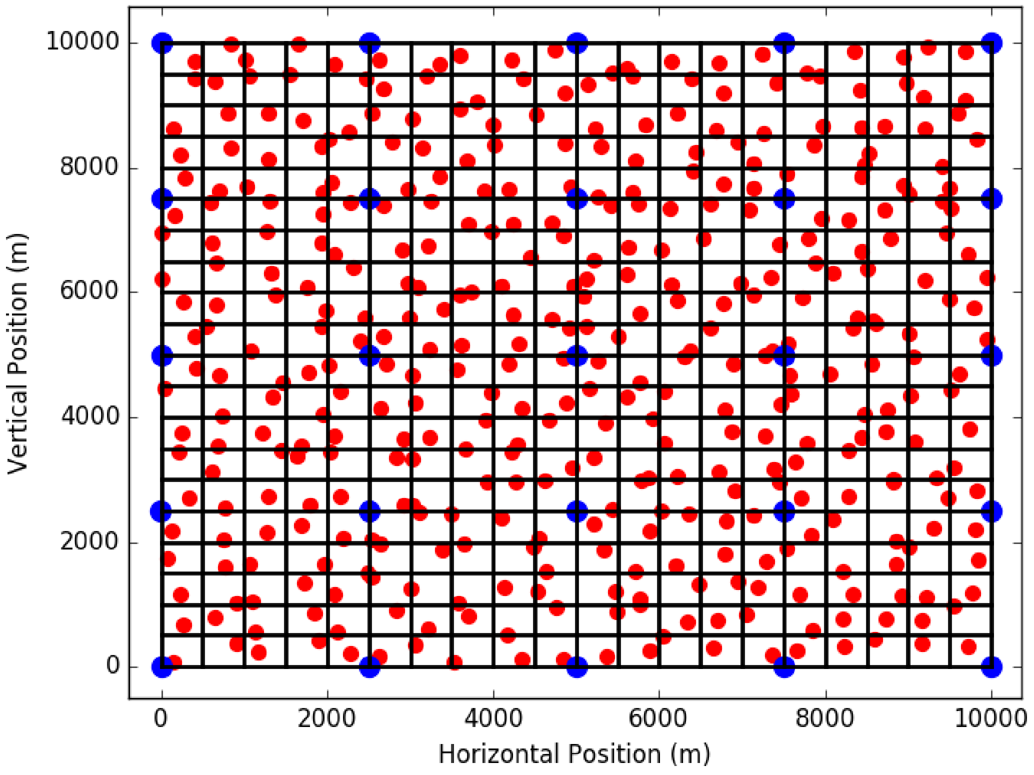


Figure 4: The hard test set. In this set we vary the position of the interference sources within their grid cells, vary the altitude of the interference sources by ± 3 m, vary the transmitter power of each interference source by ± 50W, and the sensors experience up to 2 m of positioning error.

The creation of the training and test sets was the most difficult part of this project. But thanks to the Python programming language, the process of training and testing the neural network is incredibly simple. We copy the the most important lines of code for this process in Listing 1. All of the back-propagation calculations are accomplished by the MLPClassfier methods in the scikit-learn library. Interested readers can see [11] for more information on implementing neural networks with Python and the scikit-learn library. The end result of this block of code is a trained neural network (defined as `mlp` in the code) with three hidden layers (each with 100 nodes). For brevity we have used the term jammer instead of interference source.

Listing 1: Python 2.7 code for creating, training, and using neural networks.

```
import numpy as np
from sklearn.model_selection import train_test_split #import scikit-learn packages
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier

jammers = np.load('test_data_30k.npy').item() #load pre-generated test data for m=30,000 training set
X = jammers['data'] #input vectors
y = jammers['target'] #target vectors

X_train, X_test, y_train, y_test = train_test_split(X,y, random_state=42) #split the data into training
    and test test sets

scaler = StandardScaler()
scaler.fit(10*np.log10(X_train)) #transform inputs to dBW and scale them by removing the mean and scaling
    to unit variance

X_train = scaler.transform(10*np.log10(X_train)) #scale the training and test sets
X_test = scaler.transform(10*np.log10(X_test))

mlp = MLPClassifier(hidden_layer_sizes=(100, 100, 100,), verbose=10, random_state=1, solver='adam',
    max_iter=1000) #create the pre-trained neural network

mlp.fit(X_train,y_train) #train the neural network with the training set
np.save('mlp_final_30k.npy', mlp) #save the trained neural network for additional analysis

predictions = mlp.predict(X_test) #predictions are 1 if the probability is 0.5 or higher
predictions_proba = mlp.predict_proba(X_test) #predictions as raw probabilities
predictions_log_proba = mlp.predict_log_proba(X_test) #predictions as probabilities on a log scale
```

PERFORMANCE METRICS

Next we discuss metrics for assessing the performance of the neural networks. Accuracy (A) is the strictest metric and is defined as the number of true positives (T_p) plus the number of true negatives (T_n) over the sum of the number of true positives, false positives (F_p), true negatives, and false negatives (F_n). It is very difficult for a multi-label, multi-class classifier to have high accuracy because it must exactly label the presence and absence of every class.

$$A = \frac{T_p + T_n}{T_p + F_p + T_n + F_n} \tag{10}$$

Precision (P) is defined as the number of true positives over the number of true positives plus the number of false positives. A low precision can indicate a large number of false positives.

$$P = \frac{T_p}{T_p + F_p} \tag{11}$$

Recall (R) is defined as the number of true positives over the number of true positives plus the number of false negatives. A low recall indicates many false negatives.

$$R = \frac{T_p}{T_p + F_n} \tag{12}$$

Precision and recall are are related to the (F_1) score, which is defined as the harmonic mean of precision and recall. It conveys the balance between the precision and recall.

$$F_1 = 2\frac{P \times R}{P + R} \tag{13}$$

INSPECTION OF TRAINED NEURAL NETWORK

First we can inspect the weights of the neurons in the first two layers of the neural network that we trained with an $m = 30000$ sized training set. In Figure 5 we see the weights for the neurons in the first hidden layer of the neural network. Each of the 100 plots represents a $R^{5\times5}$ matrix of weights corresponding to the input vector of 25 sensor measurements. It appears that this layer can already differentiate between different combinations of interference sources.

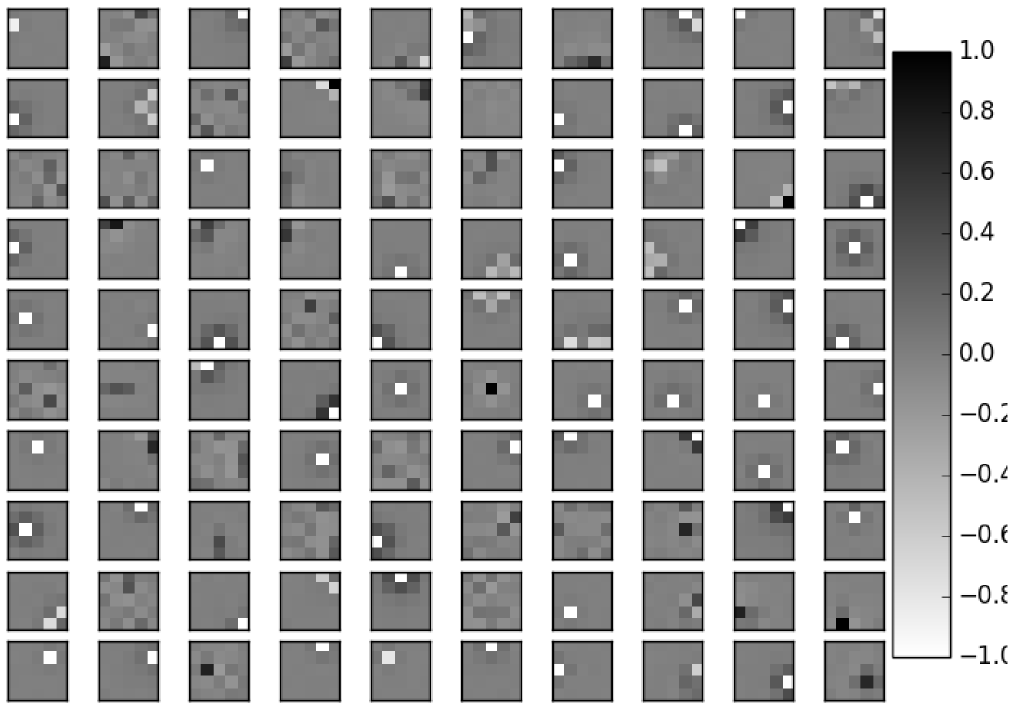


Figure 5: First hidden layer of the neural network. Each subplot represents a node in the first hidden layer of the neural network trained with an $m = 30000$ training set. The [5×5] array of pixels in each of the 100 subplots represent the trained weights associated with the 25 sensor measurements in the input vector. The weights themselves are unitless. They are mathematical constructs that link the raw input measurements to the probability that each cell of the grid contains an interference source. We only set the limits of [-1,1] in the color bar to help visualize the data. There is no limit on the range of the weights.

Now we can look at the weights in the second hidden layer as shown in Figure 6. Here we can see that each plot is now a $R^{10\times10}$ matrix since the input vector for this layer is the output of the 100 neurons in the first hidden layer.

It is less obvious which patterns this layer of the network is detecting. The third hidden layer is not shown in this paper but it displays similar characteristics. The weights of the output layer look similar but it is worth noting that an illustration of the output layer weights would contain 400 plots that each corresponded to $R^{10\times 10}$ matrices, since there are 400 cells in our grid. In other words, each of the 400 output neurons receives inputs from the 100 neurons in the third hidden layer of the network.

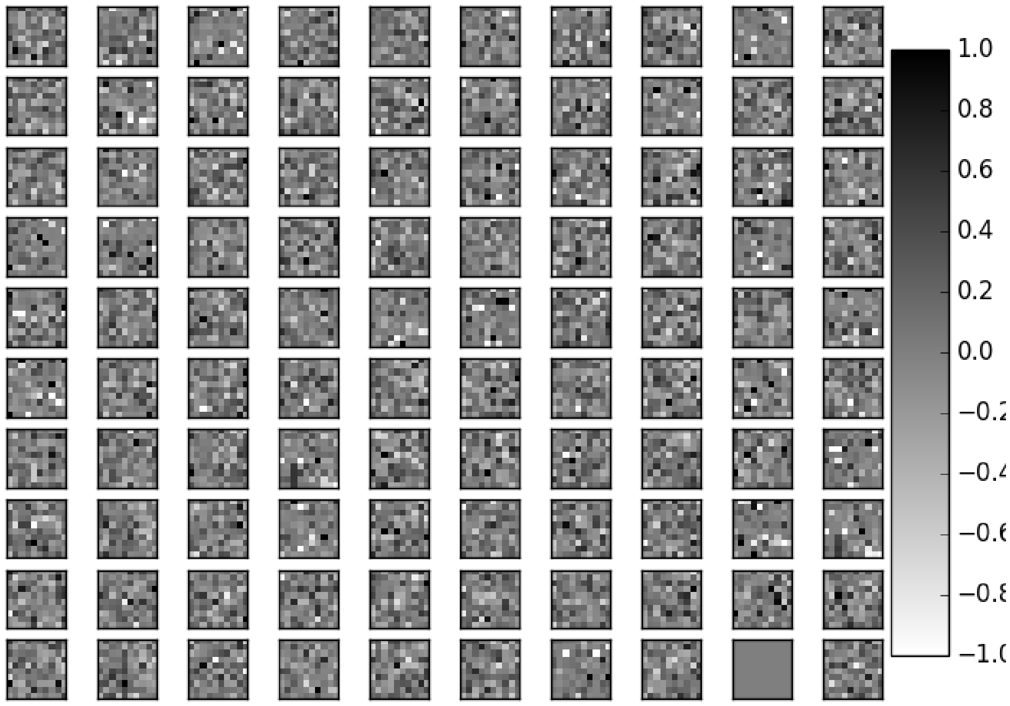


Figure 6: Second hidden layer of the neural network. Each subplot represents a node in the second hidden layer of the neural network trained with an $m = 30000$ training set. The $[10 \times 10]$ array of pixels in each of the 100 subplots represent the trained weights associated with the 100 outputs from the neurons in the first hidden layer of the network. Again, the weights themselves are unitless and we only set the limits of [-1,1] in the color bar to help visualize the data. There is no limit on the range of the weights.

PERFORMANCE ANALYSIS

Let us now consider the performance of the neural network as it is trained with different sized training sets. In Figure 7 we show the accuracy, precision, recall, and F_1 scores for the training sets of size m described in Table 2. In general, the performance of the network improves as the size of the training sets increase. As expected, the accuracy scores are significantly lower than the other three metrics. The training set improvement at $m = 300$ is likely due to over-fitting relative to the training data. The metrics for the easy and hard test sets match the overall trend for $m = 300$. An obvious area for future research is to assess the network's performance for much larger training sets.

The most useful way to visualize the benefit of this neural network framework for the simultaneous interference source localization problem is shown in Figures 8-12. These figures all show examples of 1, 2, 4, 8, and 10 interference source combinations taken from the easy test set. The top subplot in each figure shows the search grid in black, the sensor positions as blue dots, and the interference source positions as red dots. The sensor measurements subplot

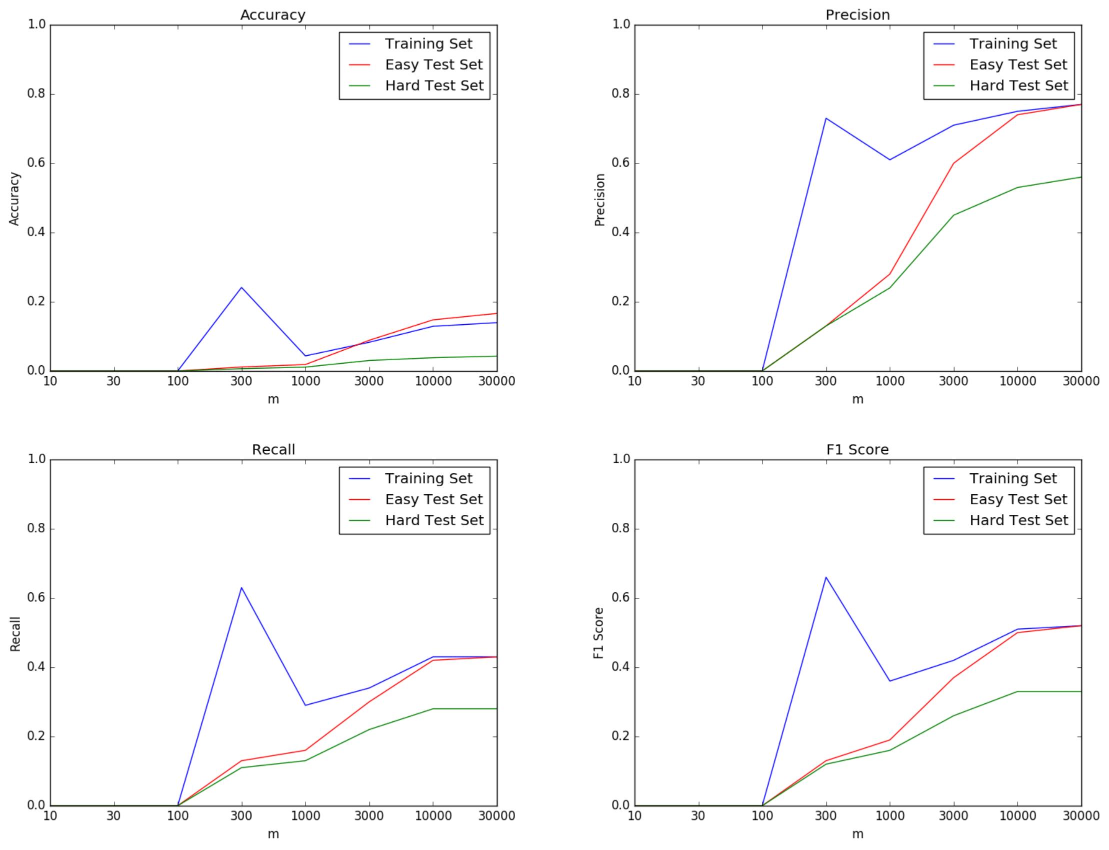


Figure 7: Accuracy, precision, recall, and F_1 scores for different sized training sets. The performance of the neural network tends to increase as it is trained with more data. As expected, accuracy is the worst performing metric. The peak at m=300 is likely due to overfitting within the training set.

visualizes the aggregate interference power measurements recorded at each sensor position. This subplot corresponds to the input vector that gets fed into the neural network. The jammer locations subplot (for brevity in the figure labeling we have again used the term jammer instead of interference source) shows the discrete representation of the interference source locations on the grid. This subplot corresponds to the truth data. The jammer probability (dB) subplot shows, on a logarithmic scale, the probability that each cell in the grid contains an interference source. This corresponds to the output of the neural network. The jammer probability subplot conveys similar information but on a probability scale of $[0, 1]$. An advantage of neural networks is that they can output probabilities rather than discrete predictions, which is not true for some of the other machine learning frameworks.

It appears that the neural network performs better for smaller combination sizes but this is likely due to the relatively small training set sizes for larger combination populations. But even with this limitation, it is impressive to see the similarities between the jammer probabilities and jammer locations subplots in Figures 11 and 12. To reiterate, we generate these predictions after the neural network with an $m = 30000$ sized training set that contained 270,400 training examples. The processing time for this training set took over 30 minutes. However, once the network was trained, it only took a matter of seconds to calculate new predictions for new input vectors. The key advantage of neural networks is that one can feed crude measurements into a pre-trained network and quickly get back useful predictions. These examples are indicative of the utility of that approach.

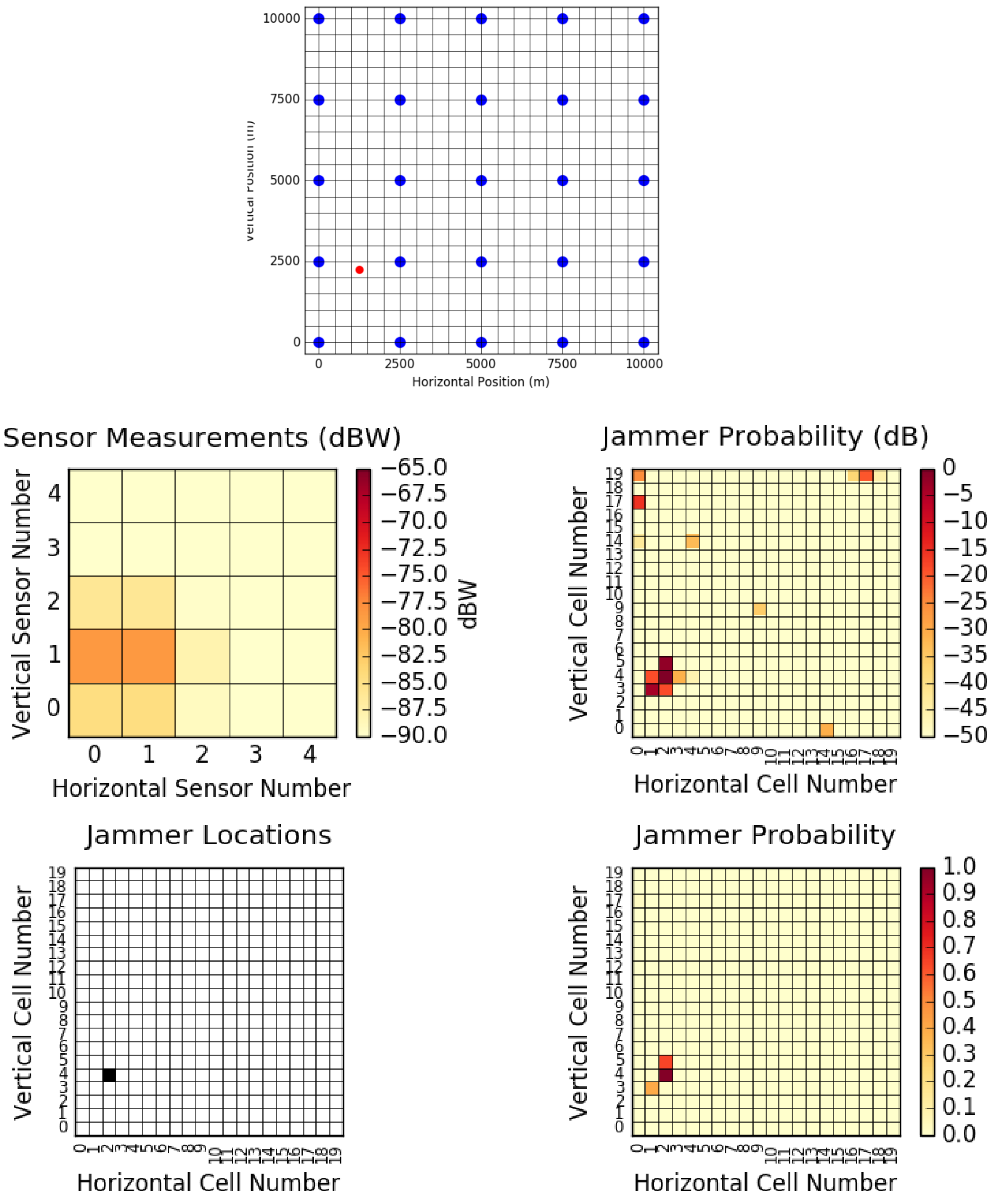


Figure 8: A single interference source. The neural network accurately assigns the cell with the highest probability of containing an interference source. Two nearby cells are indicated as well but with lower probabilities. The top subplot in each figure shows the search grid in black, the sensor positions as blue dots, and the interference source positions as red dots. The sensor measurements subplot visualizes the aggregate interference power measurements recorded at each sensor position. This subplot corresponds to the input vector that gets fed into the neural network. The jammer locations subplot shows the discrete representation of the interference source locations on the grid. This subplot corresponds to the truth data. The jammer probability (dB) subplot shows, on a logarithmic scale, the probability that each cell in the grid contains an interference source. This corresponds to the output of the neural network. The jammer probability subplot conveys similar information but on a probability scale of $[0, 1]$.

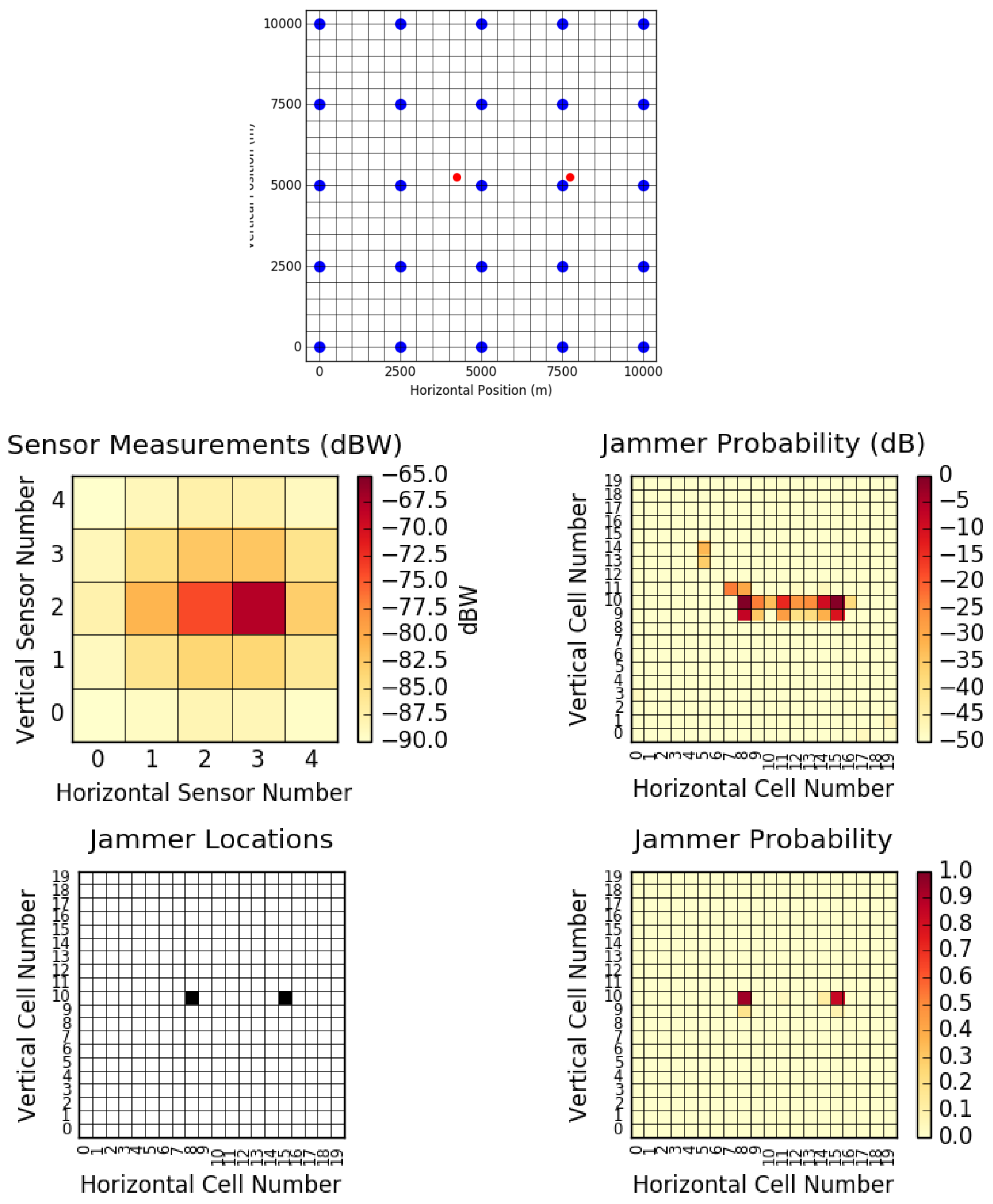


Figure 9: Two interference sources. The neural network accurately predicts the cells that contain interference sources.

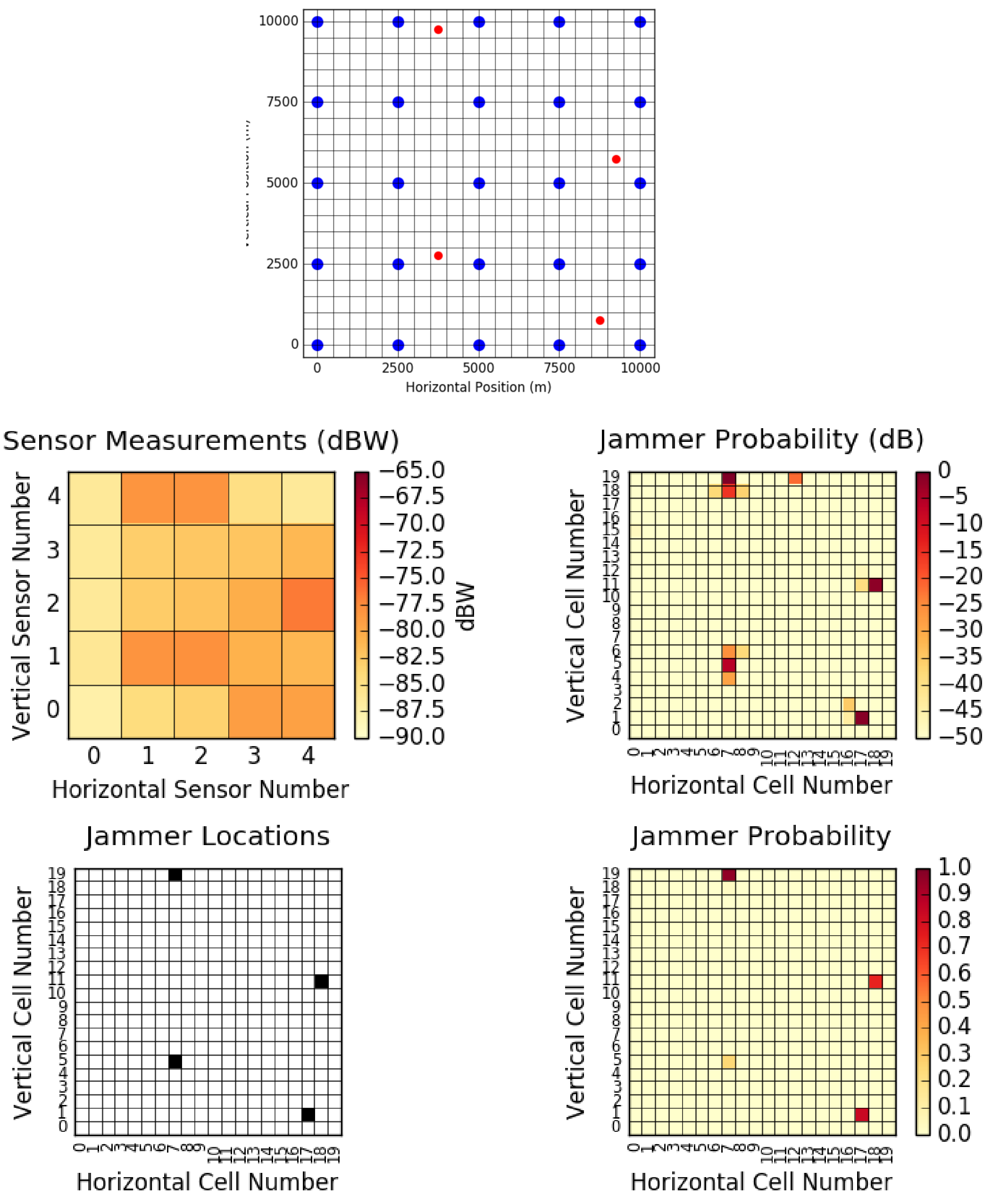
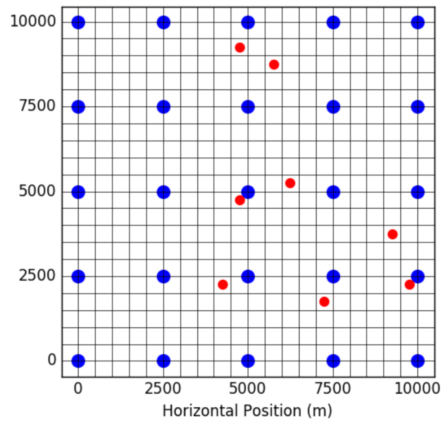


Figure 10: Four interference sources. The neural network accurately predicts the cells that contain interference sources but the probability for the bottom left cell is low, indicating that the network is not confident in this prediction.



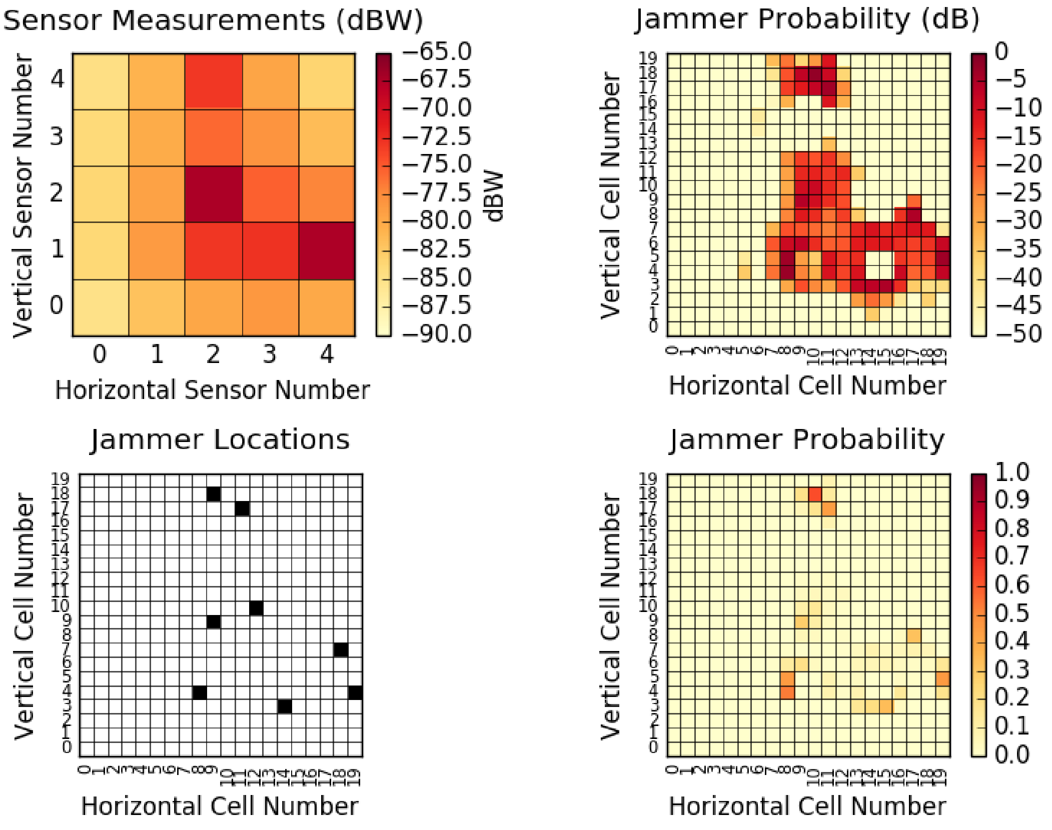


Figure 11: Eight interference sources. The neural network is beginning to predict more false positives and false negatives but the overall pattern of interference source locations is still correct.

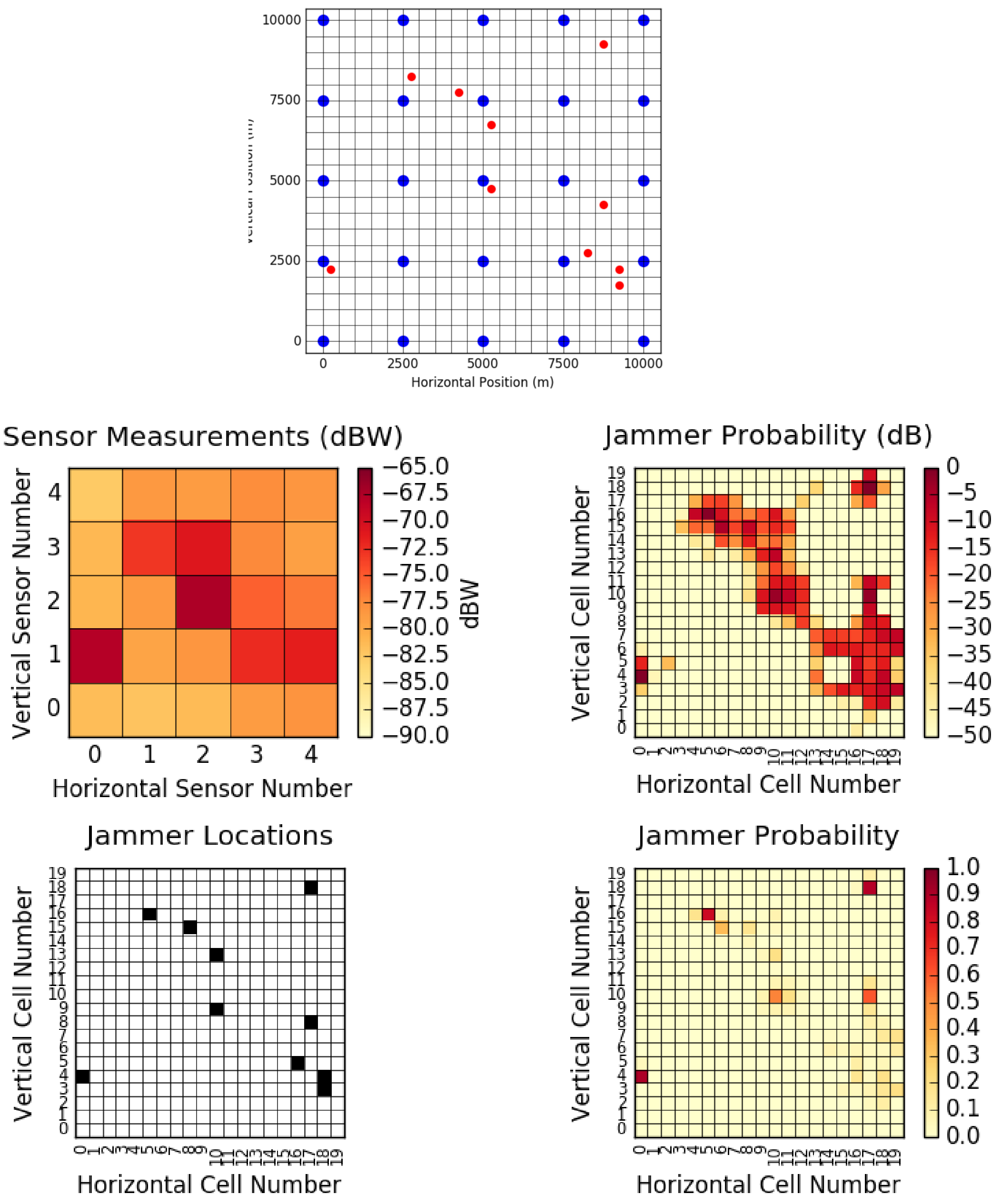


Figure 12: Ten interference sources. The neural network continues to predict more false positives and false negatives but the overall pattern of interference source locations is still correct. Also note the faint indications of interference sources in the top-center and bottom-right of the jammer probability plot. This result is particularly remarkable because the training set of just 270,400 interference source combinations is characterizing the 2.580×10^{19} possible combinations of 10 interference sources. The probability of the true positives will likely increase as the size of training sets is increased and the network is subjected to more performance optimizaiton analysis.

CONCLUSIONS AND FUTURE WORK

The goal of this paper was to present the problem of simultaneous localization of multiple interference sources in a tractable multi-label, multi-class classification framework. We found that neural networks were a viable method for implementing that framework. An important next step is to tune the macro parameters of the neural network in order to improve its performance, with the primary goal of raising the F_1 scores of the network for both the easy and hard test sets. Additional research tracks could include the application of these ideas to the classification and characterization of interference source transmitters, the use of recurrent neural networks for the localization of mobile interference sources, and real-world testbed implementations of neural networks for single and multiple interference source localization.

REFERENCES

[1] A. A. Hussein, T. A. Rahman, and C. Y. Leow, "A survey and open issues of jammer localization techniques in wireless sensor networks," *Journal of Theoretical & Applied Information Technology*, vol. 71, no. 2, 2015.

[2] M. Liu, N. Xu, and H. Li, "Multi-sensor multi-target passive locating and tracking," *International Journal of Control, Automation, and Systems*, vol. 5, no. 2, pp. 200–207, 2007.

[3] T. Cheng, P. Li, and S. Zhu, "Multi-jammer localization in wireless sensor networks," in *Computational Intelligence and Security (CIS), 2011 Seventh International Conference on*, pp. 736–740, IEEE, 2011.

[4] K. Gromov, D. Akos, S. Pullen, P. Enge, and B. Parkinson, "Gidl: generalized interference detection and localization system," *ION GPS, Salt Lake City, UT*, vol. 20, no. 0, p. 0, 2000.

[5] J. Lindström, D. Akos, O. Isoz, and M. Junered, "Gnss interference detection and localization using a network of low cost front-end modules," in *International Technical Meeting of the Satellite Division of the Institute of Navigation: 24/09/2007-28/09/2007*, pp. 1165–1172, Institute of Navigation, The, 2007.

[6] M. Trinkle and D. Gray, "Interference localisation trials using adaptive antenna arrays," in *ION GPS*, vol. 2002, p. 15th, 2002.

[7] A. K. Jain, J. Mao, and K. M. Mohiuddin, "Artificial neural networks: A tutorial," *Computer*, vol. 29, no. 3, pp. 31–34, 1996.

[8] A. Ng, "Machine learning." Online. `https://www.coursera.org/learn/machine-learning`.

[9] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[10] F. A. Administration, "Faa doubles 'blanket' altitude for many uas flights." Online, 3 2016. `https://www.faa.gov/news/updates/?newsId=85264`.

[11] scikit-learn developers, "Neural network models (supervised)." Online, 7 2017. `http://scikit-learn.org/stable/modules/neural_networks_supervised.html`.