Artificial Neural Networks for Storm Surge Predictions in NC

DHS Summer Research Team

Outline

- Introduction;
- Feedforward Artificial Neural Network;
- Design questions;
- Implementation;
- Improvements;
- Conclusions;

Brief Introduction

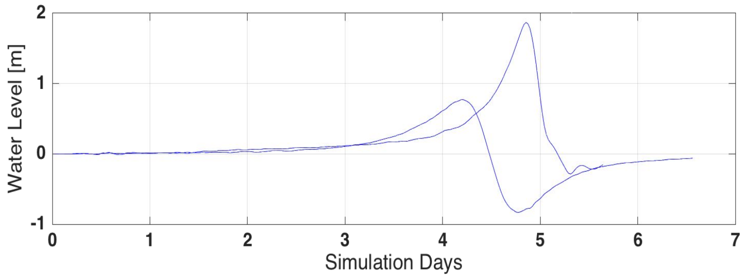
- Anton Bezuglov, Ph.D. in Computer Science and Engineering, University of South Carolina, Columbia, 2006
- Assoc. Professor of Computer Science at Benedict College
- Areas of interests: Machine learning, neural networks, algorithms, etc
- Summer Research Team 2016, sponsored by DHS
- Artificial Neural Networks for Storm Surge Prediction

Brief Introduction, contd.

- Motivation: accurate method for storm surge prediction;
- Parametric vs. Nonparametric approaches (Bishop, 2006)
- Parametric models are computationally expensive;
- Nonparametric models are cheap, but need training;
- Problem: need *large* datasets for training;
- Synthetic hurricanes;

Dataset

- 324 synthetic hurricanes;
- 193 samples per hurricane
- 6 inputs, 10 outputs
- Inputs: hurricane parameters
- Outputs: water levels at 10 locations




```
[bezuglov@ad.renci.org@bb-w540-1 code]$ cat ../data/ann_dataset_10points/track.001.dat
```

inputs

outputs

1	-3.00000	-79.000	28.090	973.4	27.150	1.100	1013.0	0.0054	0.0051	0.0056	0.0059	0.0068	0.0071	0.0114	0.0122	0.0134	0.0167
2	-2.97917	-79.000	28.130	973.4	27.190	1.100	1013.0	0.0041	0.0048	0.0056	0.0061	0.0058	0.0071	0.0100	0.0108	0.0138	0.0159
3	-2.95833	-79.000	28.170	973.4	27.230	1.100	1013.0	0.0039	0.0048	0.0055	0.0064	0.0065	0.0073	0.0083	0.0099	0.0133	0.0155
4	-2.93750	-79.005	28.205	973.4	27.260	1.095	1013.0	0.0044	0.0045	0.0056	0.0065	0.0075	0.0066	0.0079	0.0097	0.0128	0.0152
5	-2.91667	-79.010	28.240	973.4	27.290	1.090	1013.0	0.0046	0.0046	0.0058	0.0070	0.0082	0.0059	0.0091	0.0095	0.0116	0.0143
6	-2.89583	-79.010	28.280	973.4	27.330	1.090	1013.0	0.0048	0.0053	0.0062	0.0070	0.0078	0.0071	0.0094	0.0085	0.0097	0.0121
7	-2.87500	-79.010	28.320	973.4	27.370	1.090	1013.0	0.0056	0.0061	0.0066	0.0063	0.0071	0.0084	0.0087	0.0087	0.0085	0.0112
8	-2.85417	-79.015	28.355	973.4	27.405	1.090	1013.0	0.0062	0.0064	0.0066	0.0063	0.0069	0.0085	0.0093	0.0097	0.0089	0.0105
9	-2.83333	-79.020	28.390	973.4	27.440	1.090	1013.0	0.0062	0.0062	0.0063	0.0069	0.0074	0.0081	0.0112	0.0101	0.0104	0.0104
10	-2.81250	-79.020	28.430	973.4	27.475	1.090	1013.0	0.0057	0.0056	0.0064	0.0069	0.0083	0.0079	0.0119	0.0120	0.0107	0.0123
11	-2.79167	-79.020	[illegible]	973.4	27.510	1.090	1013.0	0.0047	0.0053	0.0067	0.0068	0.0082	[illegible]	0.0118	0.0140	0.0116	0.0150
12	-2.77083	-79.025	28.505	973.4	27.545	1.090	1013.0	0.0043	0.0052	0.0063	0.0065	0.0074	0.0089	0.0119	0.0138	0.0137	0.0170
13	-2.75000	-79.030	28.540	973.4	27.580	1.090	1013.0	0.0046	0.0052	0.0061	0.0064	0.0072	0.0094	0.0123	0.0124	0.0163	0.0180
14	-2.72917	-79.030	28.580	973.4	27.620	1.090	1013.0	0.0050	0.0054	0.0061	0.0065	0.0077	0.0091	0.0118	0.0126	0.0171	0.0191
15	-2.70833	-79.030	28.620	973.4	27.660	1.090	1013.0	0.0053	0.0057	0.0062	0.0067	0.0081	0.0085	0.0116	0.0129	0.0156	0.0208
16	-2.68750	-79.035	28.655	973.4	27.695	1.090	1013.0	0.0054	0.0061	0.0062	0.0065	0.0080	0.0080	0.0123	0.0128	0.0150	0.0205
17	-2.66667	-79.040	28.690	973.4	27.730	1.090	1013.0	0.0060	0.0064	0.0061	0.0061	0.0076	0.0075	0.0114	0.0131	0.0157	0.0189
18	-2.64583	-79.040	28.730	973.4	27.770	1.090	1013.0	0.0060	0.0064	0.0057	0.0060	0.0073	0.0082	0.0105	0.0119	0.0154	0.0177
19	-2.62500	-79.040	28.770	973.4	27.810	1.090	1013.0	0.0058	0.0061	0.0058	0.0065	0.0073	0.0085	0.0102	0.0104	0.0136	0.0165

Assumption

- Based on previous studies;
- Suppose input -- **x**(t), output -- **y**(t)**,** t - time;
- **x**(t) contains all information to make predictions
- **y**(t) depends on **x**(t) *only*
- **y**(t) does *not* depend on **x**(t-1), **y**(t-1), etc.

```
[bezuglov@ad.renci.org@bb-w540-1 code]$ cat ../data/ann_dataset_10points/track.001.dat
1   -3.00000   -79.000   28.090   973.4   27.150   1.100   1013.0   0.0054   0.0051  0.0056   0.0059  0.0068   0.0071  0.0114   0.0122  0.0134   0.0167
2   -2.97917   -79.000   28.130   973.4   27.190   1.100   1013.0   0.0041   0.0048  0.0056   0.0061  0.0058   0.0071  0.0100   0.0108  0.0138   0.0159
3   -2.95833   -79.000   28.170   973.4   27.230   1.100   1013.0   0.0039   0.0048  0.0055   0.0064  0.0065   0.0073  0.0083   0.0099  0.0133   0.0155
4   -2.93750   -79.005   28.205   973.4   27.260   1.095   1013.0   0.0044   0.0045  0.0056   0.0065  0.0075   0.0066  0.0079   0.0097  0.0128   0.0152
5   -2.91667   -79.010   28.240   973.4   27.290   1.090   1013.0   0.0046   0.0046  0.0058   0.0070  0.0082   0.0059  0.0091   0.0095  0.0116   0.0143
6   -2.89583   -79.010   28.280   973.4   27.330   1.090   1013.0   0.0048   0.0053  0.0062   0.0070  0.0078   0.0071  0.0094   0.0085  0.0097   0.0121
7   -2.87500   -79.010   28.320   973.4   27.370   1.090   1013.0   0.0056   0.0061  0.0066   0.0063  0.0071   0.0084  0.0087   0.0087  0.0085   0.0112
8   -2.85417   -79.015   28.355   973.4   27.405   1.090   1013.0   0.0062   0.0064  0.0066   0.0063  0.0069   0.0085  0.0093   0.0097  0.0089   0.0105
9   -2.83333   -79.020   28.390   973.4   27.440   1.090   1013.0   0.0062   0.0062  0.0063   0.0069  0.0074   0.0081  0.0112   0.0101  0.0104   0.0104
10   -2.81250   -79.020   28.430   973.4   27.475   1.090   1013.0   0.0057   0.0056  0.0064   0.0069  0.0083   0.0079  0.0119   0.0120  0.0107   0.0123
11   -2.79167   -79.020   28.470   973.4   27.510   1.090   1013.0   0.0047   0.0053  0.0067   0.0068  0.0082   0.0084  0.0118   0.0140  0.0116   0.0150
12   -2.77083   -79.025   28.505   973.4   27.545   1.090   1013.0   0.0043   0.0052  0.0063   0.0065  0.0074   0.0089  0.0119   0.0138  0.0137   0.0170
13   -2.75000   -79.030   28.540   973.4   27.580   1.090   1013.0   0.0046   0.0052  0.0061   0.0064  0.0072   0.0094  0.0123   0.0124  0.0163   0.0180
14   -2.72917   -79.030   28.580   973.4   27.620   1.090   1013.0   0.0050   0.0054  0.0061   0.0065  0.0077   0.0091  0.0118   0.0126  0.0171   0.0191
15   -2.70833   -79.030   28.620   973.4   27.660   1.090   1013.0   0.0053   0.0057  0.0062   0.0067  0.0081   0.0085  0.0116   0.0129  0.0156   0.0208
16   -2.68750   -79.035   28.655   973.4   27.695   1.090   1013.0   0.0054   0.0061  0.0062   0.0065  0.0080   0.0080  0.0123   0.0128  0.0150   0.0205
17   -2.66667   -79.040   28.690   973.4   27.730   1.090   1013.0   0.0060   0.0064  0.0061   0.0061  0.0076   0.0075  0.0114   0.0131  0.0157   0.0189
18   -2.64583   -79.040   28.730   973.4   27.770   1.090   1013.0   0.0060   0.0064  0.0057   0.0060  0.0073   0.0082  0.0105   0.0119  0.0154   0.0177
19   -2.62500   -79.040   28.770   973.4   27.810   1.090   1013.0   0.0058   0.0061  0.0058   0.0065  0.0073   0.0085  0.0102   0.0104  0.0136   0.0165
```

Outline

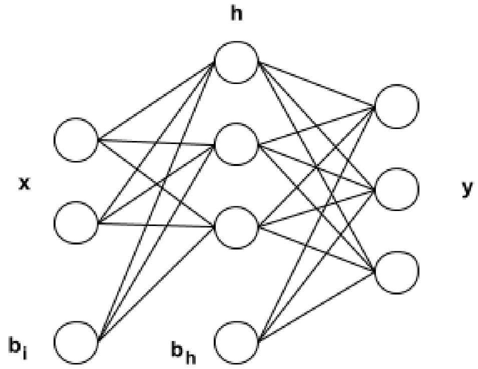
- Introduction;
- Feedforward Artificial Neural Network;
- Design questions;
- Implementation;
- Improvements;
- Conclusions;

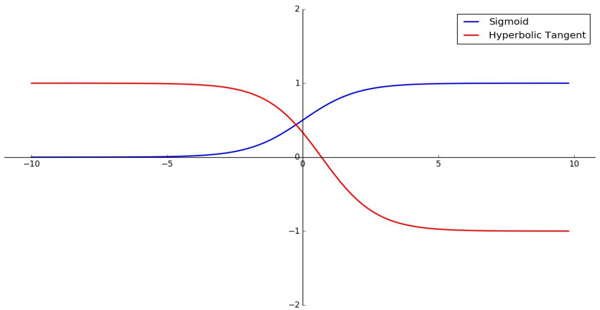
Regression with a FF ANN

- Problem: find a function f(.), so that:
- $\mathbf{y}_p = f(\mathbf{x})$, $\mathbf{y}_p$-- storm surge predictions
- f(.) -- can be a Feed Forward Artificial Neural Network (FF ANN);
- Train FF ANN to minimize the error between $\mathbf{y}$ and $\mathbf{y}_p$
- Use synthetic storms to train;

FF ANN's

- One hidden layer ANN, two layer model;
- Information travels from left to right;
- Nodes are variables (inputs, outputs, and hidden);
- Edges -- independent parameters;
- Nonlinear function;
- Complexity determined by # of multiplications
- Approx. $O(N^2)$, N - # of hidden nodes
- Backpropagation algorithm





$$f(\mathbf{x}) = W_h * \mathbf{h} + \mathbf{b_h}$$

$$\mathbf{h} = \sigma(W_i * \mathbf{x} + \mathbf{b_i})$$

Design Questions

- Architecture?
- Number of hidden layers?
- Size of each layer?
- Choice of nonlinear function?
- Initial weights/biases?
- Learning rate?
- Learning rate decay?
- Algorithm for training?
- Clipping gradients?
- Dealing with overfitting?
- Loss function?

Design Questions, contd.

- Architecture? -- *Two hidden layer multiple outputs*
- Number of hidden layers? -- *two hidden layers*
- Size of each layer? -- *16-64 neurons, second layer larger*
- Choice of nonlinear function? -- *TanH*
- Initial weights/biases? -- *N(0, 0.01)*
- Learning rate? -- *0.001 -- 0.01*
- Learning rate decay? -- *0.5*
- Algorithm for training? -- *ADAM optimization algorithm*
- Clipping gradients? -- *yes, 1.25-1.5 norm*
- Dealing with overfitting? -- *validation set, 15%*
- Loss function-- *Mean Squared Error (MSE)*

Design Questions, contd.

- Stochastic optimization
  - Use portions of the training dataset: batches
  - Training dataset: 228 storms, batches: 19, 57, 114
  - Or Training dataset: 225 storms, batches: 3, 5, 9, 15, 45, 225
- Inputs normalization
  - Inputs vary by 2-3 orders of magnitude
  - Too long to converge
  - Calculate moments for each input param in the training dataset
  - Normalize inputs
  - Store the moments along with the model

Design Summary

- Split dataset into training (70%), validation (15%), and testing (15%);
- Two hidden layer FF ANN ($N_1 < N_2$, less inputs than outputs);
- Train to minimize MSE;
- Check for overfitting on the validation dataset;
- Evaluate performance on the testing dataset;

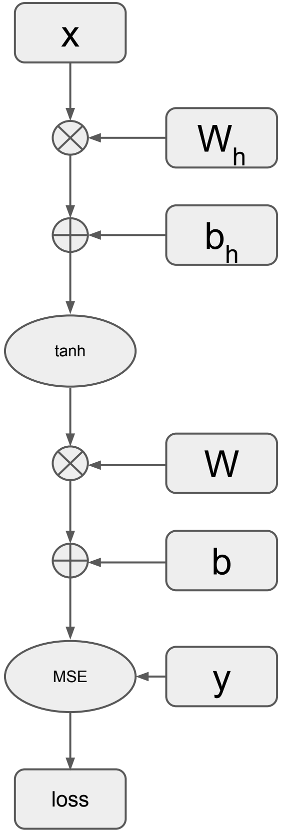
Outline

- Introduction;
- Feedforward Artificial Neural Network;
- Design questions;
- Implementation;
- Improvements;
- Conclusions;

Implementation: TensorFlow

- TensorFlow -- Open Source Library for Machine Intelligence;
- Algorithms are graphs, nodes -- operations, edges -- tensors

```
tf_train_dataset = tf.placeholder(tf.float32, shape=(batch_size, 6)) #train_dataset2.shape(2)
tf_train_labels = tf.placeholder(tf.float32, shape=(batch_size, 2))
tf_valid_dataset = tf.constant(valid_dataset2)
tf_test_dataset = tf.constant(test_dataset2)

weights_0 = tf.Variable(tf.truncated_normal([6,hidden_nodes_1], dtype = tf.float32))
biases_0 = tf.Variable(tf.zeros([hidden_nodes_1], dtype = tf.float32))

weights_1 = tf.Variable(tf.truncated_normal([hidden_nodes_1,hidden_nodes_2], dtype = tf.float32))
biases_1 = tf.Variable(tf.zeros([hidden_nodes_2], dtype = tf.float32))

weights_2 = tf.Variable(tf.truncated_normal([hidden_nodes_2,2], dtype = tf.float32))
biases_2 = tf.Variable(tf.zeros([2], dtype = tf.float32))


input_layer_output = tf.sigmoid(tf.matmul(tf_train_dataset, weights_0) + biases_0)
hidden_layer_output = tf.sigmoid(tf.matmul(input_layer_output, weights_1) + biases_1)
#hidden_layer_output = tf.nn.dropout(hidden_layer_output, 0.5)
hidden_layer_output = tf.matmul(hidden_layer_output, weights_2) + biases_2


loss = tf.cast(tf.reduce_mean(tf.reduce_mean(tf.square(hidden_layer_output-tf_train_labels))),tf.float32)
#loss = tf.cast(tf.reduce_mean(tf.reduce_mean(tf.square(tf.square(hidden_layer_output-tf_train_labels)))),tf.float32

global_step = tf.Variable(0.00, trainable=False)
learning_rate = tf.train.exponential_decay(starter_learning_rate, global_step, num_steps, 0.96, staircase=False)
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss, global_step=global_step)
```



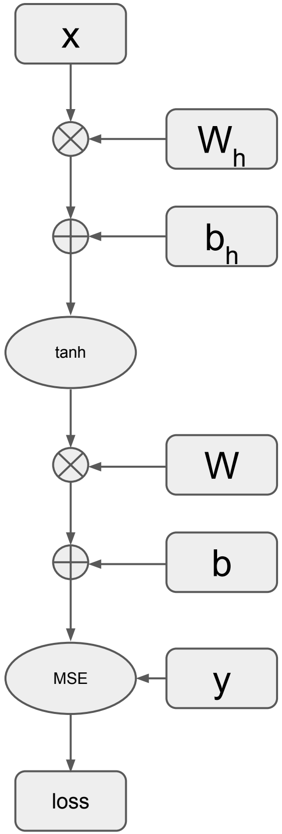
Implementation: Training and Evaluation

- Graph variables can be evaluated/called
- To train -- call optimizer variable
- To evaluate -- call loss variable
- etc.

```
batch_data = train_dataset2[offset:(offset + batch_size), :]
batch_output = train_output[offset:(offset + batch_size), :]
feed_dict = {tf_train_dataset : batch_data, tf_train_labels : batch_output}
_, l, predictions = session.run([optimizer, loss, train_prediction],feed_dict=feed_dict)
```

List of graph variables to evaluate

Inputs for placeholders



Implementation: Dealing with Gradients

Calculate gradients

Clip

Apply gradients

Evaluate train_op to perform a single train iteration

```
# Training portion of the graph
# Eval train_op to perform one step training to minimize loss
#===================================================
# Prepare global step and learning rate for optimization
global_step = tf.get_variable(
    'global_step', [],
    initializer=tf.constant_initializer(0), trainable=False)
learning_rate = tf.train.exponential_decay(
    FLAGS.learning_rate, global_step, FLAGS.max_steps,
    FLAGS.learning_rate_decay, staircase=False)

 # Create ADAM optimizer
optimizer = tf.train.AdamOptimizer(learning_rate)

# Calculate gradients and apply them
grads, v = zip(*optimizer.compute_gradients(loss))
grads, _ = tf.clip_by_global_norm(grads, 1.25)
apply_gradient_op = optimizer.apply_gradients(zip(grads,v), global_step = global_step)

# Smoothen variables after gradient applications
variable_averages = tf.train.ExponentialMovingAverage(
    FLAGS.moving_avg_decay, global_step)
variables_averages_op = variable_averages.apply(tf.trainable_variables())
train_op = tf.group(apply_gradient_op, variables_averages_op)

init = tf.initialize_all_variables()
sess = tf.Session(config = tf.ConfigProto(
    allow_soft_placement = False, # allows to utilize GPU's & CPU's
    log_device_placement = False)) # shows GPU/CPU allocation
```

Implementation: Multiple GPU’s

- Each GPU has same graph but individual inputs/outputs;
- Calculate gradients on each GPU;
- Average gradient;
- Apply gradients;
- Update graphs;

Algorithm 1 Training algorithm for a multiple output ANN on parallel GPU’s using TensorFlow

procedure TRAIN($\mathbf{x}, \hat{\mathbf{y}}$) ▷ $\mathbf{x}$ – inputs, $\hat{\mathbf{y}}$ – observations
  $\mathbf{x_n}$ = Normalize($\mathbf{x}$) ▷ subtract means and divide by std component-wise
  for g in GPUs **do**
    $g \leftarrow InitializeNetwork()$ ▷ Assign a copy of the network on each GPU
  end for
  for e in Epochs **do**
    $b_{1..GPUs} \leftarrow GetDataBatch(\mathbf{x_n}, \hat{\mathbf{y}}, size)$ ▷ Fetch batches of training data for each GPU
    $g_{1..GPUs} \leftarrow CalculateGradients(b_{1..GPUs})$ ▷ Calculate gradients on each GPU
    $g \leftarrow Mean(g_{1..GPUs})$ ▷ Find average gradient
    $\mathbf{v} \leftarrow AdamOptimizer(g, rate)$ ▷ Obtain the new values of network parameters
    UpdateNetworks($\mathbf{v}$) ▷ Propagate changes to all networks
  end for
end procedure

Implementation: Restore ANN

- Save model: *weights, biases, and input moments*;
- Train/Run modes;
- Train -- open file, train ANN, save ANN;
- Run -- open file, open model, run, save outputs;
- Train, approx. 1-20 minutes;
- Run, 0.11 sec (324x193 samples);

```
[bezuglov@ad.renci.org@bb-w540-1 code]$ python ./ilt_default_feed.py
Processed 0/324

Processed 100/324

Processed 200/324

Processed 300/324

inputs: calculate new means, stds for dataset with 44004 samples
outputs: calculate new means, stds for dataset with 44004 samples
normalizing inputs and outputs
inputs: using provided means, stds for dataset with 9264 samples
outputs: using provided means, stds for dataset with 9264 samples
normalizing inputs and outputs
inputs: using provided means, stds for dataset with 9264 samples
outputs: using provided means, stds for dataset with 9264 samples
normalizing inputs and outputs
Num hurricanes in train 228, validation 48, test 48
Step 0 (83.63 op/sec): Training MSE: 0.22264, Validation CC: 0.0051, MSE: 0.12320
Step 2500 (229.94 op/sec): Training MSE: 0.02879, Validation CC: 0.8781, MSE: 0.02115
Step 5000 (236.97 op/sec): Training MSE: 0.01371, Validation CC: 0.9163, MSE: 0.01313
Step 7500 (220.66 op/sec): Training MSE: 0.01001, Validation CC: 0.9320, MSE: 0.01122
Step 10000 (201.46 op/sec): Training MSE: 0.00859, Validation CC: 0.9402, MSE: 0.01026
Step 12500 (213.17 op/sec): Training MSE: 0.00767, Validation CC: 0.9437, MSE: 0.00956
Step 15000 (215.19 op/sec): Training MSE: 0.00697, Validation CC: 0.9496, MSE: 0.00867
Step 17500 (221.53 op/sec): Training MSE: 0.00651, Validation CC: 0.9487, MSE: 0.00874
Step 20000 (221.29 op/sec): Training MSE: 0.00623, Validation CC: 0.9505, MSE: 0.00838
Step 22500 (214.64 op/sec): Training MSE: 0.00595, Validation CC: 0.9509, MSE: 0.00822
Step 25000 (215.57 op/sec): Training MSE: 0.00571, Validation CC: 0.9526, MSE: 0.00804
Step 27500 (211.51 op/sec): Training MSE: 0.00562, Validation CC: 0.9527, MSE: 0.00799
Step 30000 (205.55 op/sec): Training MSE: 0.00543, Validation CC: 0.9538, MSE: 0.00795
Step 32500 (210.79 op/sec): Training MSE: 0.00541, Validation CC: 0.9540, MSE: 0.00782
Step 35000 (210.35 op/sec): Training MSE: 0.00526, Validation CC: 0.9552, MSE: 0.00782
Step 37500 (222.17 op/sec): Training MSE: 0.00515, Validation CC: 0.9555, MSE: 0.00780
Step 40000 (214.22 op/sec): Training MSE: 0.00503, Validation CC: 0.9555, MSE: 0.00761
Step 42500 (212.13 op/sec): Training MSE: 0.00494, Validation CC: 0.9548, MSE: 0.00764
Step 45000 (212.13 op/sec): Training MSE: 0.00486, Validation CC: 0.9563, MSE: 0.00766
Step 47500 (211.01 op/sec): Training MSE: 0.00475, Validation CC: 0.9565, MSE: 0.00750
Step 50000 (214.69 op/sec): Training MSE: 0.00469, Validation CC: 0.9564, MSE: 0.00759
Training summary:
Test MSE: 0.00769
Location 0: CC: 0.9511, MSE: 0.002691
Location 1: CC: 0.9616, MSE: 0.002060
Location 2: CC: 0.9713, MSE: 0.001207
Location 3: CC: 0.9751, MSE: 0.000948
Location 4: CC: 0.9670, MSE: 0.003165
Location 5: CC: 0.9334, MSE: 0.005563
Location 6: CC: 0.9678, MSE: 0.009057
Location 7: CC: 0.9377, MSE: 0.025910
Location 8: CC: 0.9284, MSE: 0.010701
Location 9: CC: 0.9369, MSE: 0.015614
```

```
[bezuglov@ad.renci.org@bb-w540-1 code]$ python ./ilt_default_feed.py
Model ./models/save_two_layers_32_64_15000_AB/model.ckpt-50000 restored
Elapsed time: 0.11 sec.
Outputs saved as ./test_track_out2.dat
[bezuglov@ad.renci.org@bb-w540-1 code]$
```

FF ANN: Performance

- Two hidden layer FF ANN (32,64)

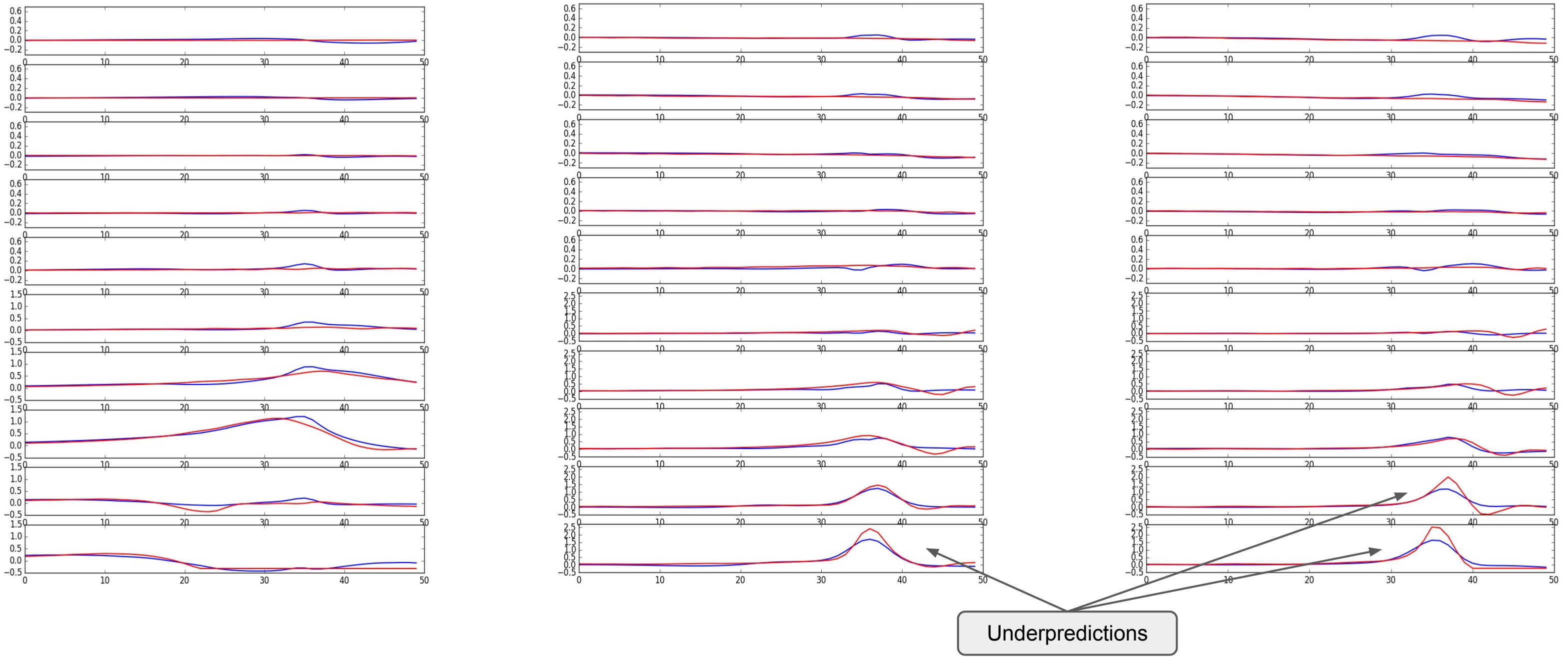
			Before and after landfall	Landfall only		
Location	MSE	R	$P(\lvert e\rvert \leq 0.1m)$	$e^*, P(\lvert e\rvert \leq e^*) = 0.95$	$P(\lvert e\rvert \leq 0.1m)$	$P(\lvert e\rvert \leq 0.5m)$
1	0.0017	0.963	0.966	0.15	0.864	0.998
2	0.0012	0.975	0.976	0.13	0.909	0.998
3	0.0008	0.988	0.992	0.10	0.950	0.999
4	0.0004	0.992	0.993	0.10	0.953	0.999
5	0.0014	0.976	0.978	0.17	0.861	0.994
6	0.0038	0.932	0.935	0.32	0.692	0.985
7	0.0079	0.892	0.900	0.46	0.531	0.960
8	0.0112	0.853	0.864	0.51	0.477	0.949
9	0.0095	0.901	0.910	0.44	0.566	0.960
10	0.0175	0.833	0.850	0.49	0.475	0.953

“Easy”

“Difficult”

FF ANN: Performance

ADCIRC
FF ANN

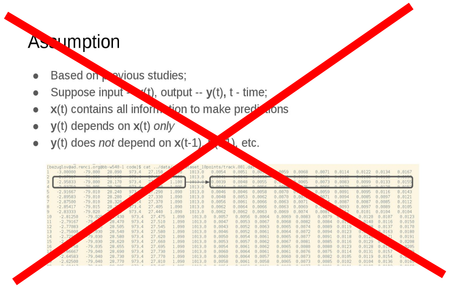


FF ANN: Summary

- Multi-output ANN: one model for several locations
- MSE's are approx. 0.006 m^2
- CC's are 0.95
- ANN has no error *before* and *after* the storm surge;
- Larger errors at storm surge;
- Low MSE's b/c of zeros;

Does **y**(t) depend on **x**(t) *and* something else?

Does **x**(t) miss information?



Acknowledgements

- Many thanks to Brian Blanton, Ph.D.
- RENCI, CRC
- DHS SRT Program

This research was performed under an appointment to the U.S. Department of Homeland Security (DHS) Science & Technology (S&T) Directorate Office of University Programs Summer Research Team Program for Minority Serving Institutions, administered by the Oak Ridge Institute for Science and Education (ORISE) through an interagency agreement between the U.S. Department of Energy (DOE) and DHS. ORISE is managed by ORAU under DOE contract number DE-AC05-06OR23100. All opinions expressed in this presentation are the author’s and do not necessarily reflect the policies and views of DHS, DOE or ORAU/ORISE.