

Data Science:
Machine Learning
(MTH 9899)
Baruch College
Lecture 2

Miguel A. Castro

Today we'll cover:

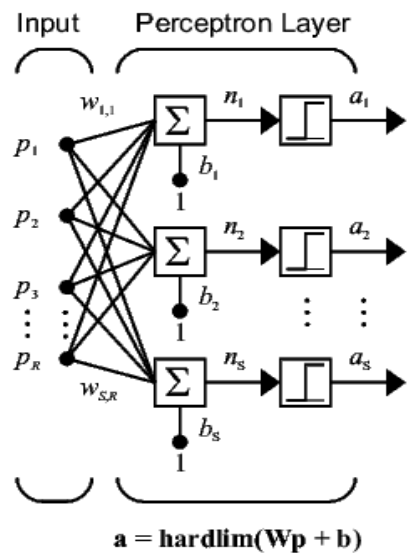
- Pop Quiz (20% of your grade)... *April Fool's Joke!*
- Review of Last Lecture
- Continue Neural Networks
- Motivate Bayesian Methods
- Case Study: House Values in Boston
- Start on Classification and ROC Curves

Last Time...

- Class Syllabus, HW1, Projects...
 - Questions?
- Importance/Usefulness of Machine Learning in light of “Big Data” “Big Computing” and “Automated Information Extraction.” The bigger the data, the more automation makes sense.
- Brief History of ML in the Context of AI.
- Definition of ML: Automated way of extracting useful information from large data sets.
- Paradigms: Supervised (target provided) vs. Unsupervised (target not provided) Learning.
- Neural Networks
 - Important and useful examples of ML.
 - Original motivation was as models of the biological brain.
 - Useful and a vast field in their own right.
 - Good starting point for ML.

Single Perceptron (Review)

- Perceptron
 - Hard Limit Activation Functions (thought to emulate biological neurons)
- Single-Layer Perceptron



R inputs

S outputs

\mathbf{W} is an $S \times R$ matrix of weights

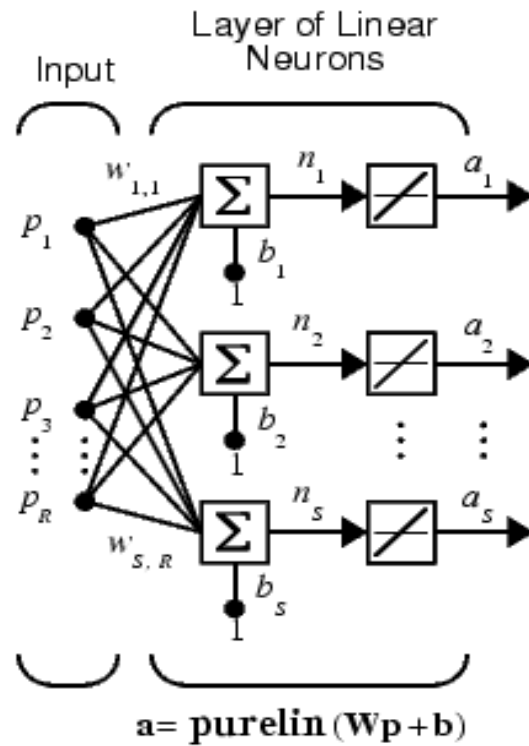
\mathbf{b} is an S vector of biases

- Perceptron Learning Rule.
- A Single Perceptron correctly Classifies AND, OR functions, but not XOR.
- A Single Perceptron can only handle Linearly Separable Classes.

Multi-Layer Perceptrons (Review)

- Multi-Layer Perceptrons (MLP)
 - Can be constructed to *implement* XOR (in a contrived way; not useful).
 - *In principle*, can solve Linearly Non-Separable problems using a Hidden Layer; **but...**
 - There is no analog of Single-Perceptron Learning Rule for Multiple Layers.
 - Need differentiable Activation Functions to produce Automated Learning Rule.

ADALINES/MADALINES (Review)



Linear Activations

R inputs

S outputs

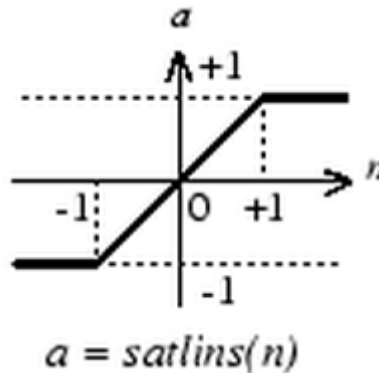
\mathbf{W} is an $S \times R$ matrix of weights

\mathbf{b} is an S vector of biases

- Linear Activation Functions: Differentiable → Can use gradient Descent to produce the **Delta Learning Rule**.
- Delta Learning Rule can be automated for multiple layers.
- Linear Activations still only work for **Linearly Separable** Problems.

SATALINES (Review)

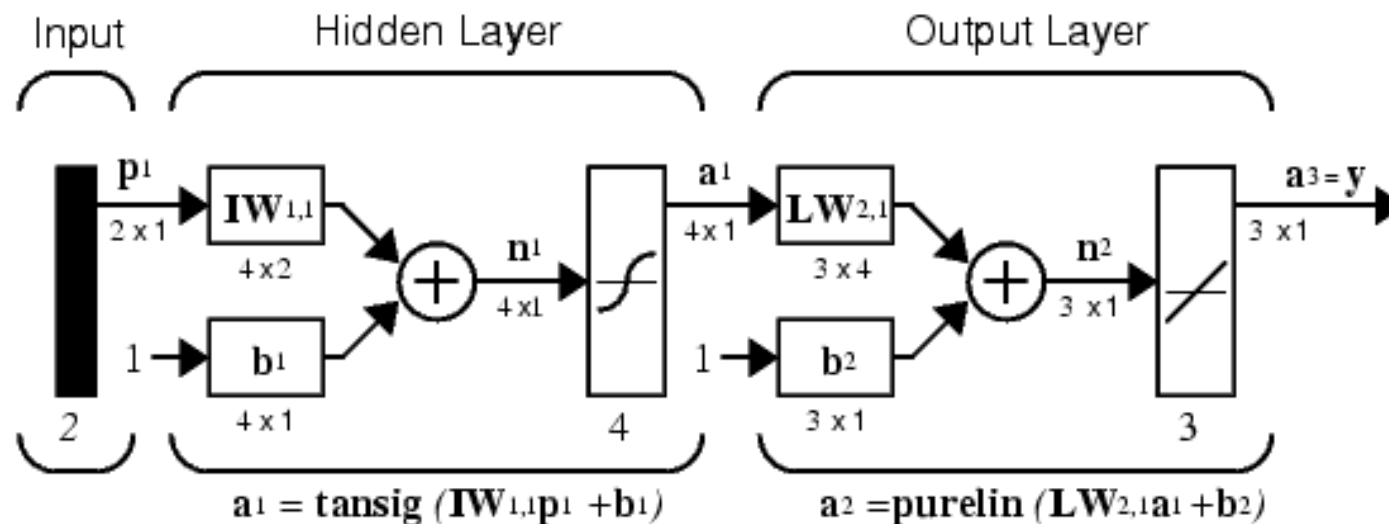
- Saturated Linear Transfer Functions in the Hidden Layer:



- Combine features of Perceptrons and ADALINEs.
- Differentiable in the linear region: Can use Delta Learning Rule for automated training.
- Can solve XOR and more general LNS Problems.
- Can show that with only 2 satlins Hidden Nodes and a Linear Output node will solve XOR automatically with Delta Learning Rule. **Exercise.**

Feedforward Neural Networks (Review)

- Wish List for General-Purpose Approximator is to Combine:
 - Multiple Layers (we learned from Perceptrons this is needed for LNS problems).
 - Require Automated Learning Rule → Smooth (Differentiable) Activations.
 - Nonlinear Activations also needed (we learned from ADALINEs and SATALINEs).
- Enter FeedForward Neural Networks (FFNN):



- Multi-Layer
- Outputs don't loop back into inputs.
- Sigmoidal or other non-linear but smooth (differentiable) Hidden-Layer Activations.
- Learning Rule: **Backpropagation**: A generalization of the Delta Learning Rule.
- This is a **Universal Approximator**. Hornik *et.al.* (1989).

Universal Approximation (Review)

- FeedForward Neural Networks:

- Also known as MLPs with sigmoidal activations, but we won't use that name here (MLPs is reserved for Hard-Limit Activation Multi-Layer Perceptrons).
- A Single Hidden-Layer of Non-Linear Activations is sufficient for universal approximation (White *et al.*).
- The most common architecture (the workhorse) is Sigmoidal Hidden Layers and a Linear Output Layer.

- Backpropagation

- Generalization of the Delta Learning Rule.
- It's highly parallelizable.
- Perform a gradient descent by starting at the output and propagating the error backwards to the input weights using the chain rule of differentiation:

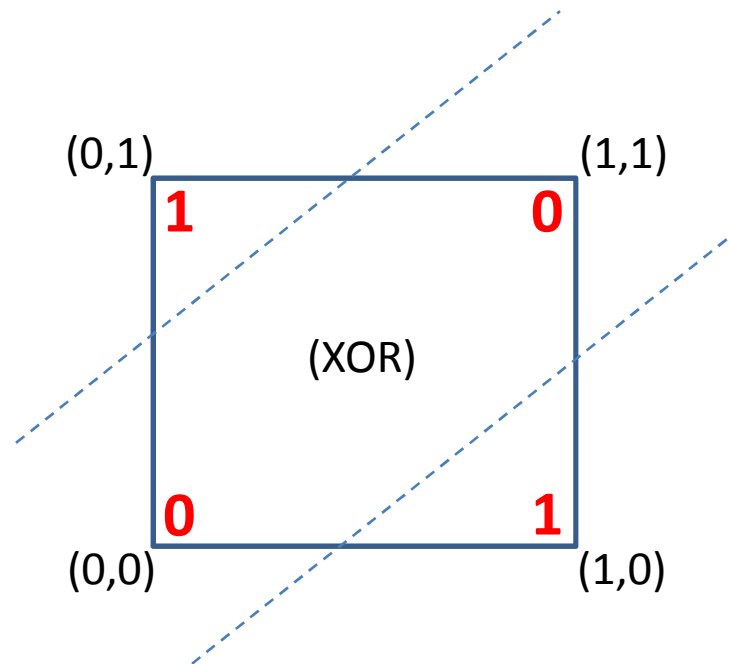
$$\Delta w_{hj} = -\eta \frac{\partial E}{\partial w_{hj}} = -\eta \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial a_h} \frac{\partial a_h}{\partial w_{hj}} \dots$$

Universal Approximation (Review)

- FFNNs can solve a range of problems:
 - Function Approximation
 - Clustering
 - Time Series Forecasting
 - Classification
- Criteria for Universal Approximation:
 - At least one Hidden Layer (*i.e.*, at least one layer in-between input and output layers).
 - Non-linear Activations in the hidden layer(s)
- Criterion for Automated Learning Rule:
 - Smooth (differentiable) Activations throughout in order to exploit gradient-descent type learning rules such as Backpropagation.

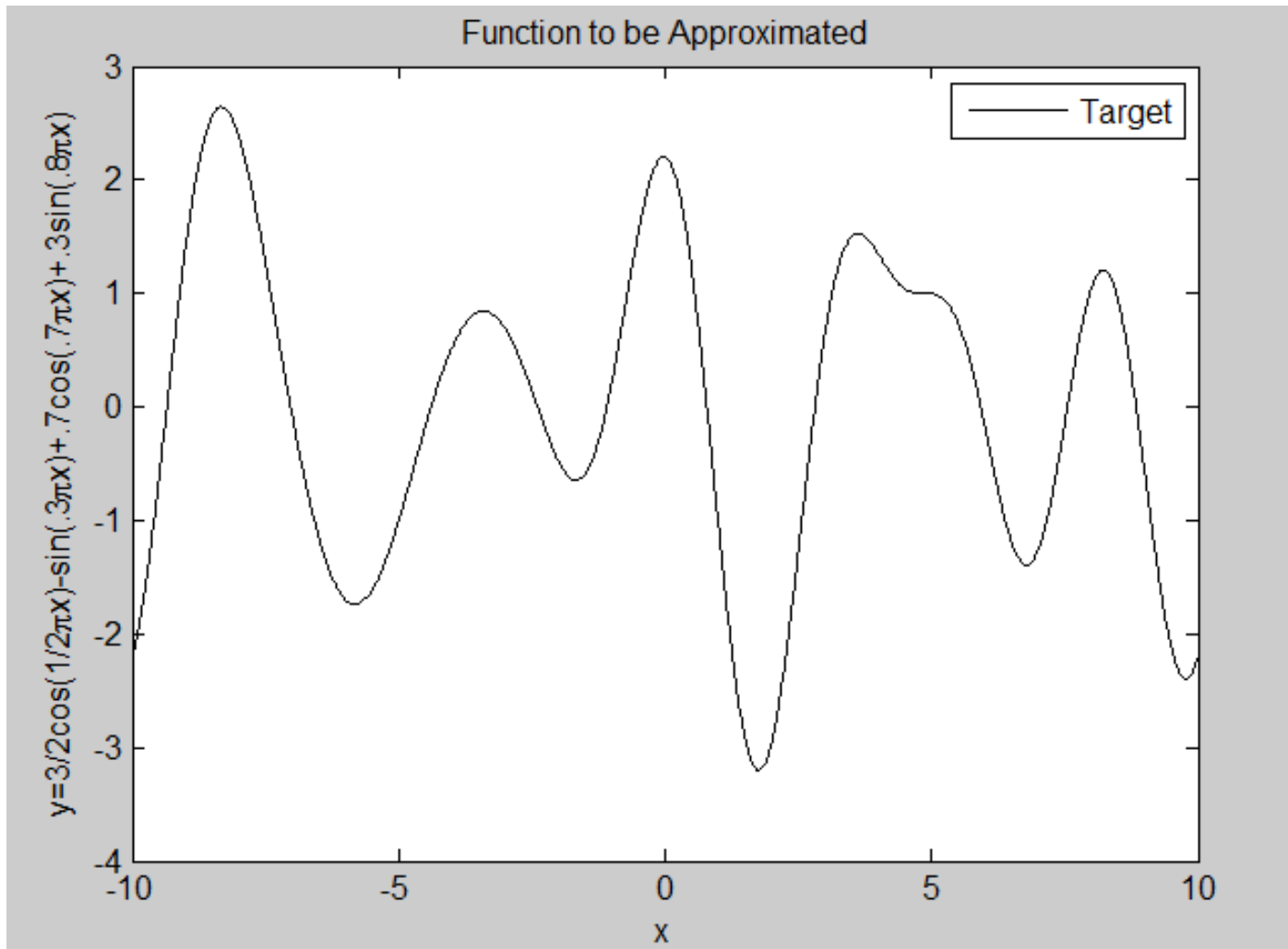
Demos from Last Time...

- FFN with single hidden layer of non-linear (sigmoidal) activations and linear output can classify XOR, where Perceptrons and ADALINEs Flounder.



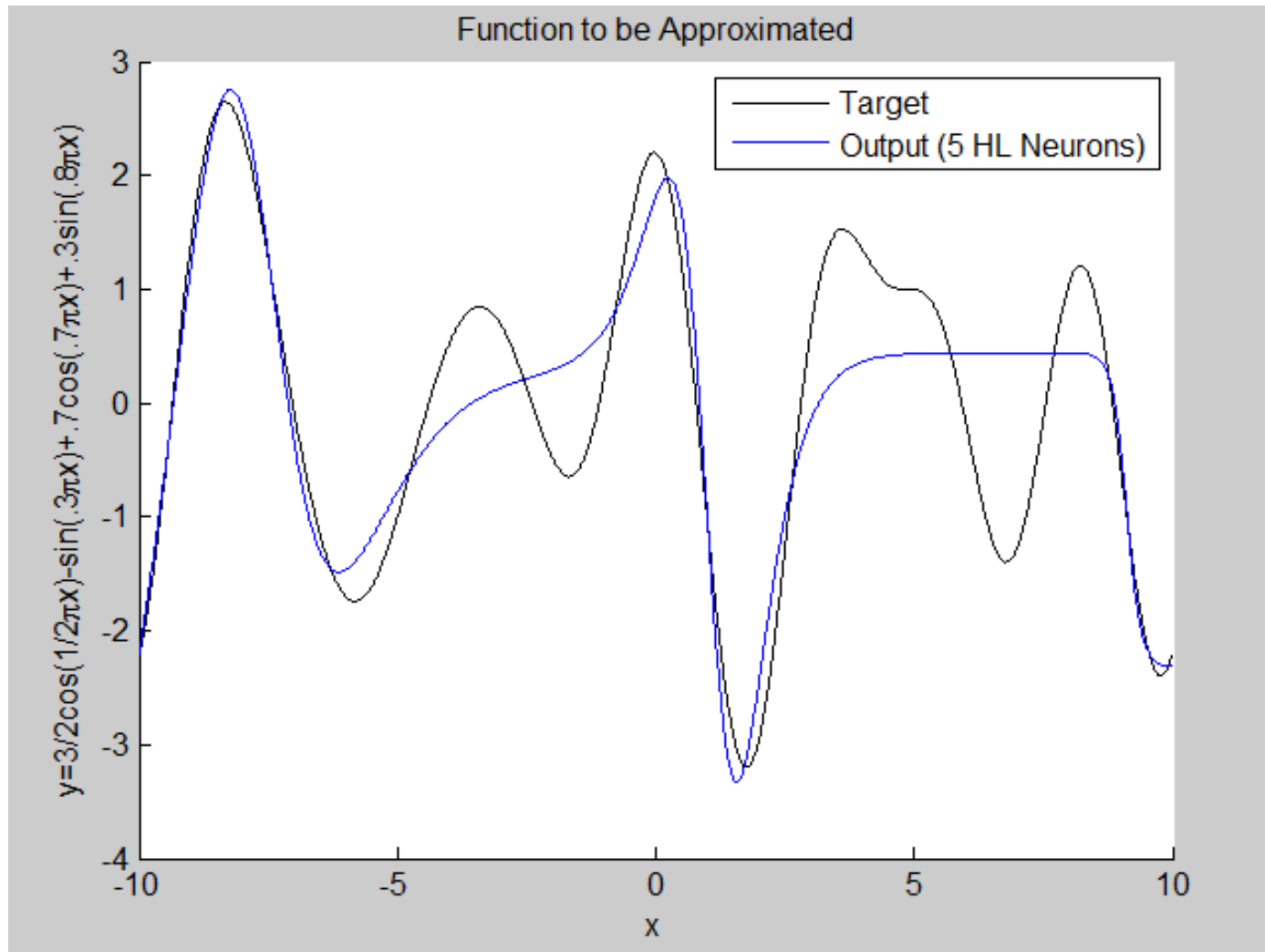
- Can classify any **Linearly Non-Separable** (LNS) Problem, even more difficult ones in higher dimensions, or with more than 2 classes.
- On to Function Approximation...

Function Approximation



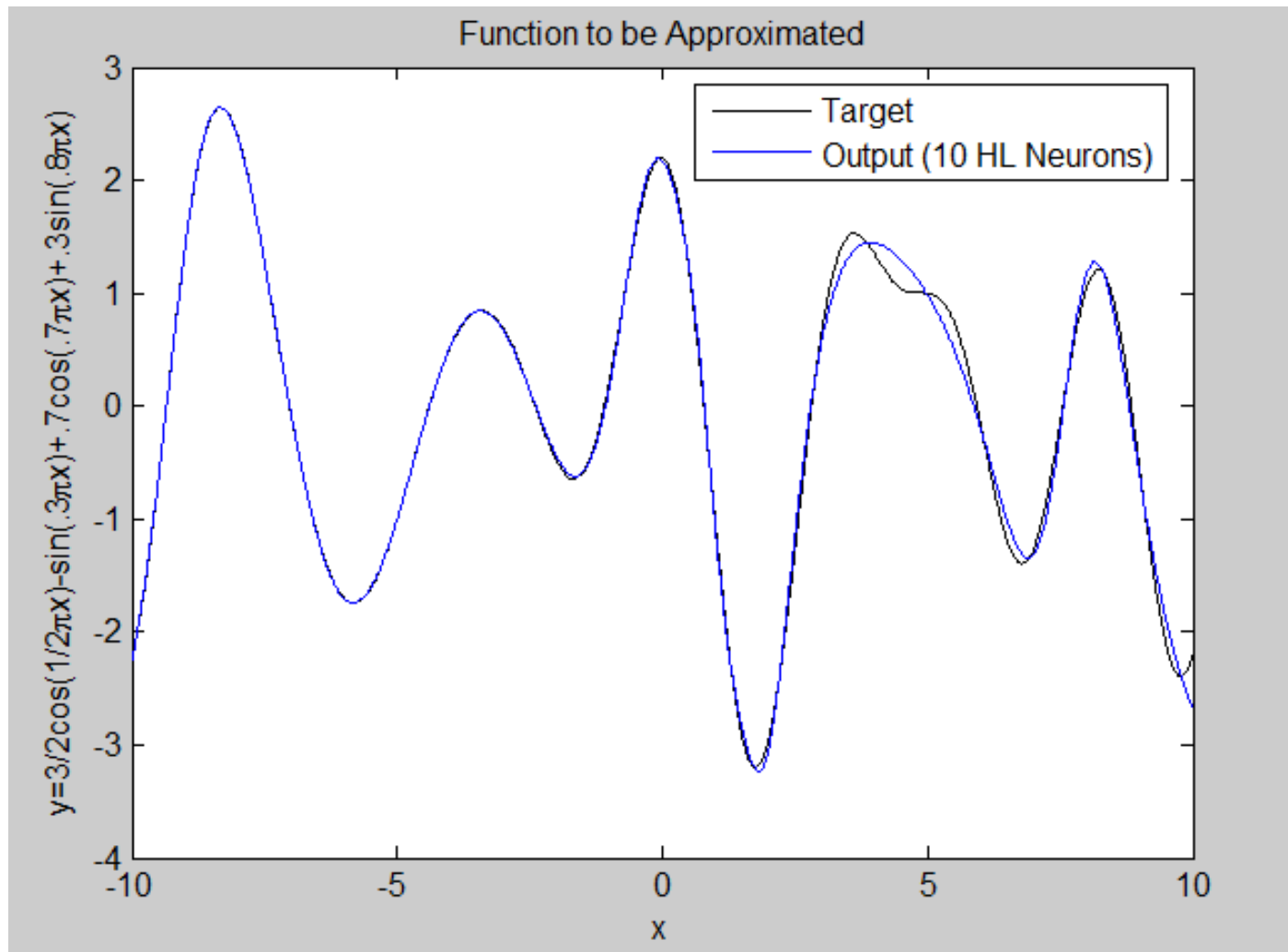
(...)

Function Approximation (FNN, 5 HN)



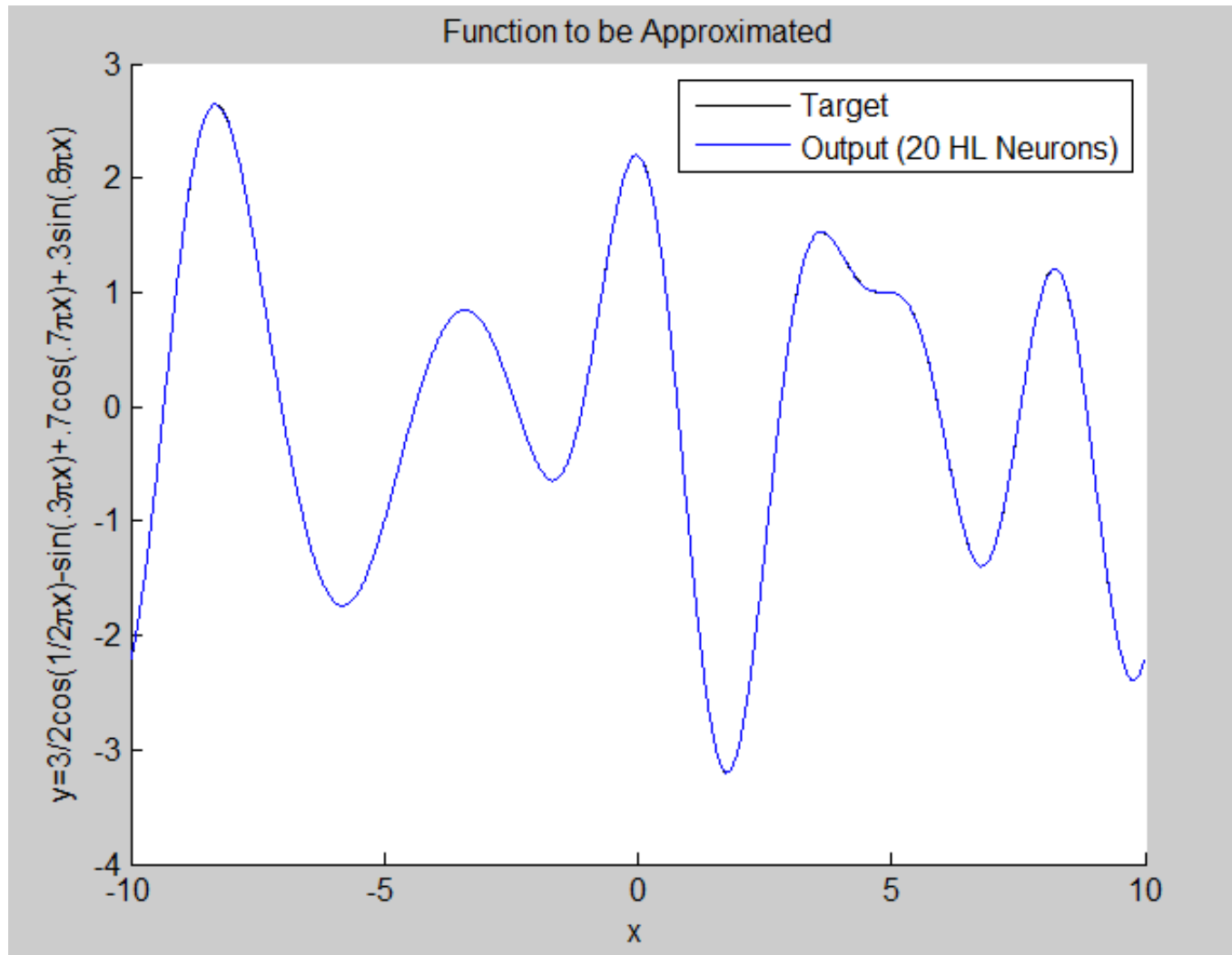
- Pretty Good

Function Approximation (FNN, 10 HN)



- Better

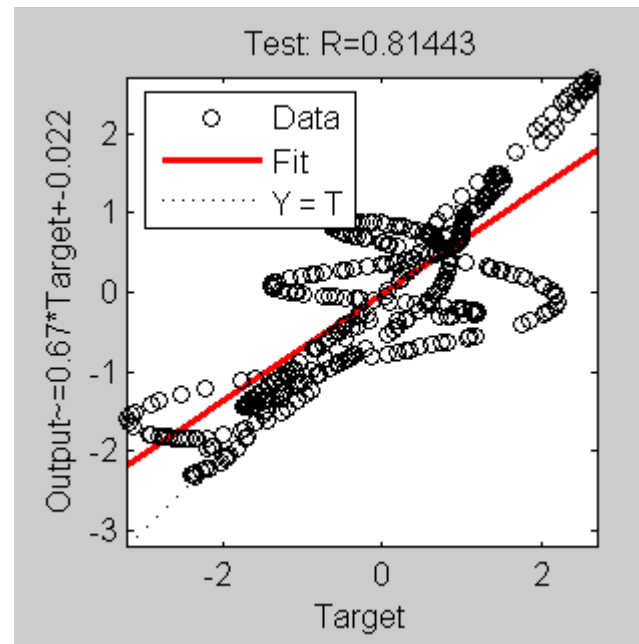
Function Approximation (FNN, 20 HN)



- Spot on!

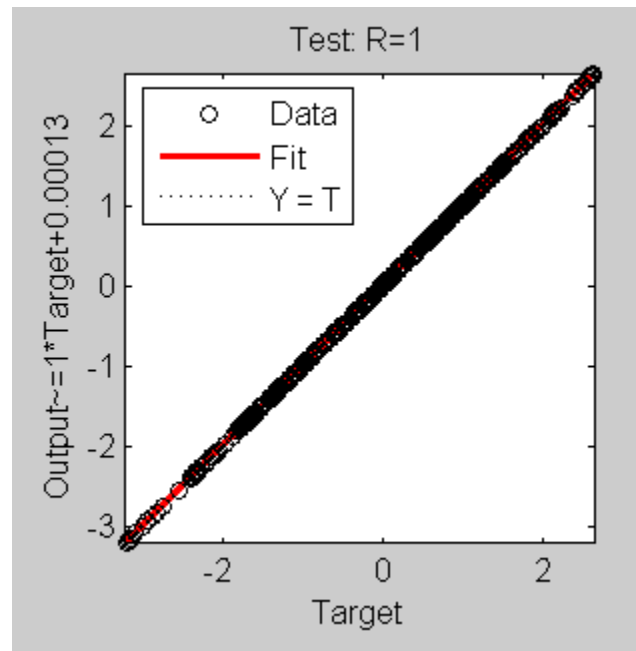
Function Approximation: Evaluation

- We don't have to rely on eyeballing.
- Regress Output vs. Target to get Goodness-of-Fit statistics such as Correlation, R^2 , t -stats...:



5 Hidden-Layer Neurons

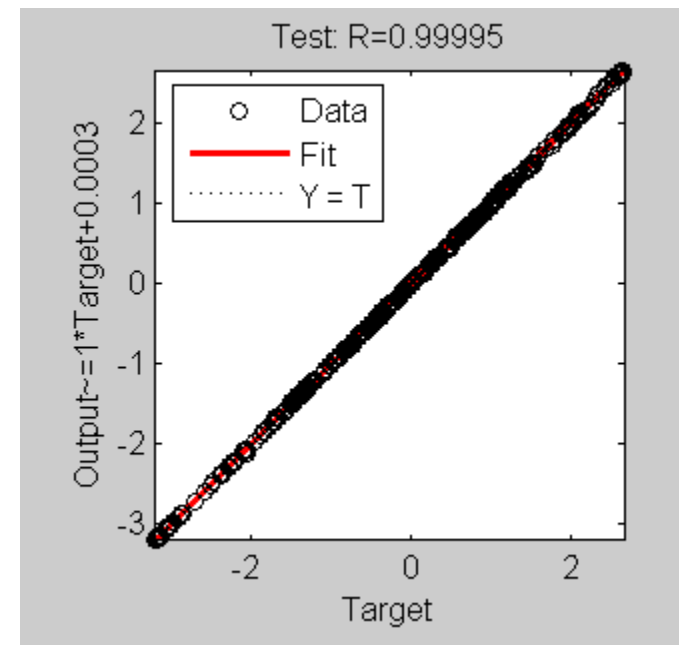
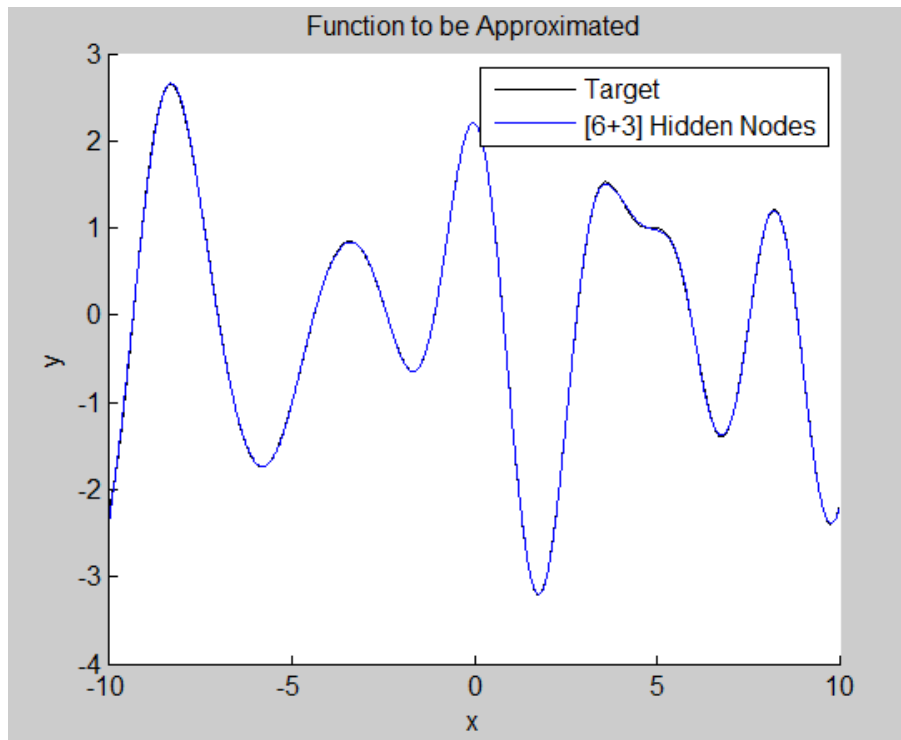
Function Approximation: Evaluation



20 Hidden-Layer Neurons

Single vs. Multiple Hidden Layers

- We found that a single hidden layer with 20 nodes (neurons) worked extremely well.
- Now Consider two hidden layers with [6+3] nodes:

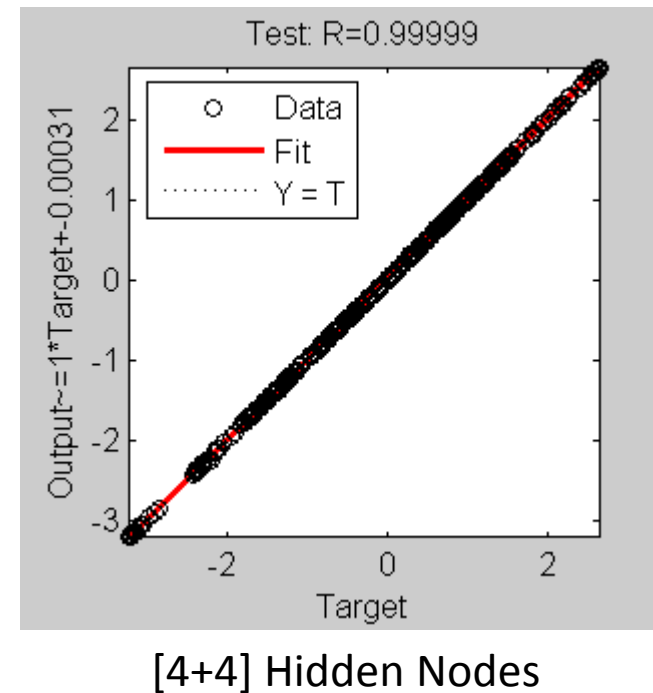
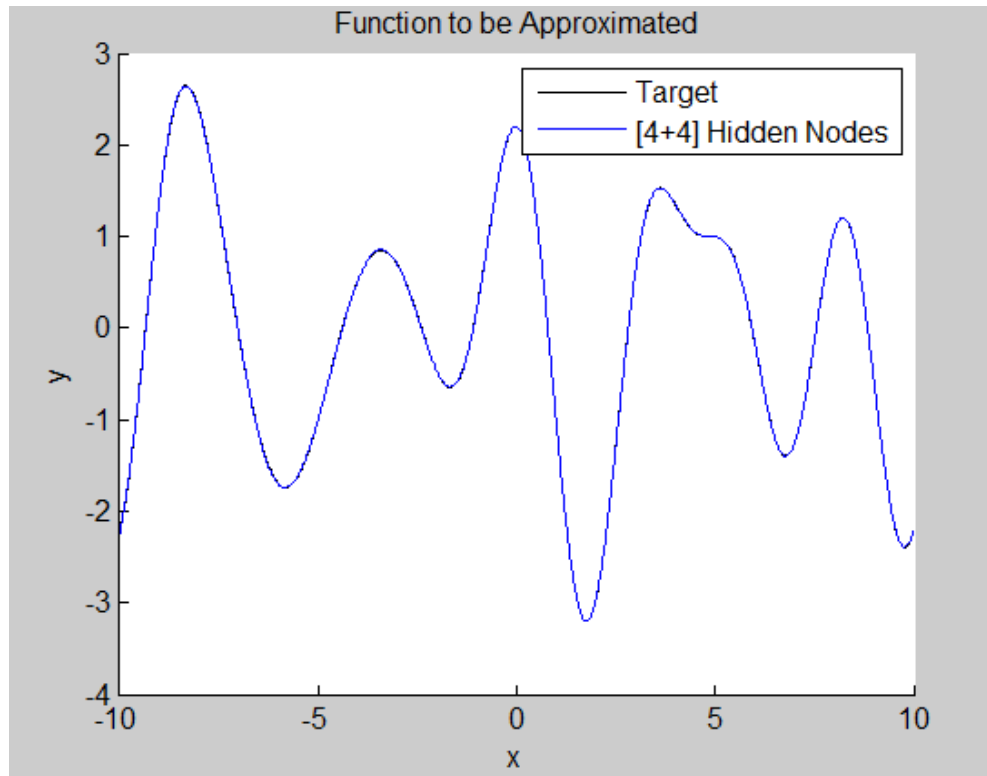


[6+3] Hidden Nodes

- Works extremely well also; 9 vs. 20 total Hidden Nodes (more parsimonious?).

Single vs. Multiple Hidden Layers

- Now Consider two hidden layers with [4+4] nodes:



- Works extremely well also (8 vs. 20 Hidden Nodes).

Single vs. Multiple Hidden Layers

- Let's take a closer look.
- Are Neural Networks Parametric or Nonparametric techniques?

- They can be treated as Nonparametric:

$$Y_i = f(\mathbf{X}_i) + \varepsilon_i,$$

- Where $f \in \mathcal{F}$, some class of functions.
- The only requirement is for \mathcal{F} to be sufficiently rich, but we're not really interested in the *form* of f itself.
- So, to the extent that NNs are “black boxes” they behave as if they were Nonparametric. However...

Single vs. Multiple Hidden Layers

- NNs have internal *parameters*: the connection strengths (weights and biases).
- In this regard they are Parametric, and suffer from the need for parsimony exhibited by Parametric models.
- Consider a FFNN with I Inputs, J Hidden Layers, and O outputs, with H_j nodes in the j^{th} Hidden Layer.
- The number of parameters (weights) is just:

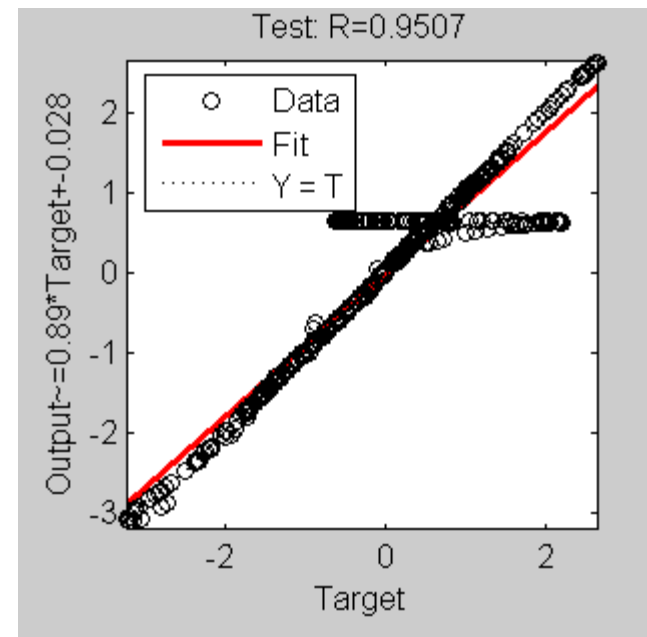
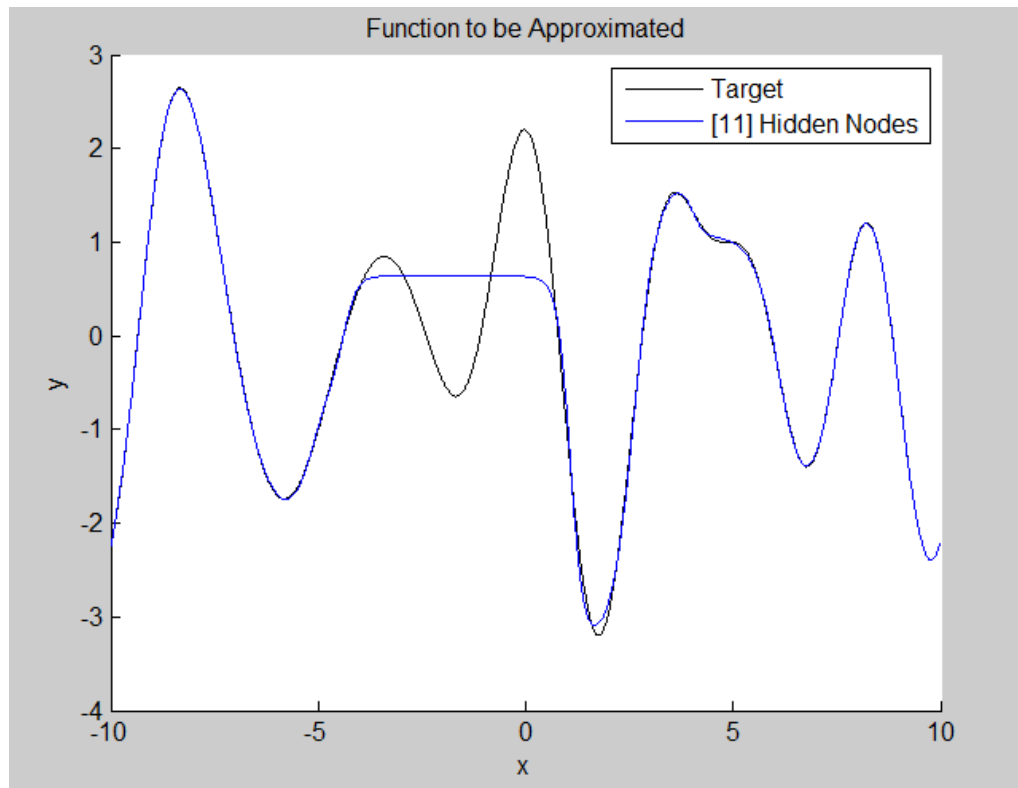
$$\mathcal{N} = (I + 1)H_1 + \sum_{\substack{j=1 \\ (J>1)}}^{J-1} (H_j + 1)H_{j+1} + (H_J + 1)O.$$

Single vs. Multiple Hidden Layers

- From this, we see that the [20] architecture above (single hidden layer with 20 nodes) had $\mathcal{N} = 61$, while the [6+3] architecture (two hidden layers with 6 and 3 nodes, respectively) had $\mathcal{N} = 37$, and the [4+4] architecture had $\mathcal{N} = 33$ internal parameters.
- All of these architectures worked quite well with correlations very close to 1 (full approximation).
- However, they have very different numbers of degrees of freedom (weights).
- On the other hand, the [10] architecture had $\mathcal{N} = 31$. It didn't work quite as well, but had a similar \mathcal{N} to the double-layer architectures.
- Notice that if we had used an [11] architecture, we would have $\mathcal{N} = 34$, which is close to the \mathcal{N} for [4,4]. Let's try this...

Single vs. Multiple Hidden Layers

- Now Consider [11] architecture (one HL with 11 nodes):

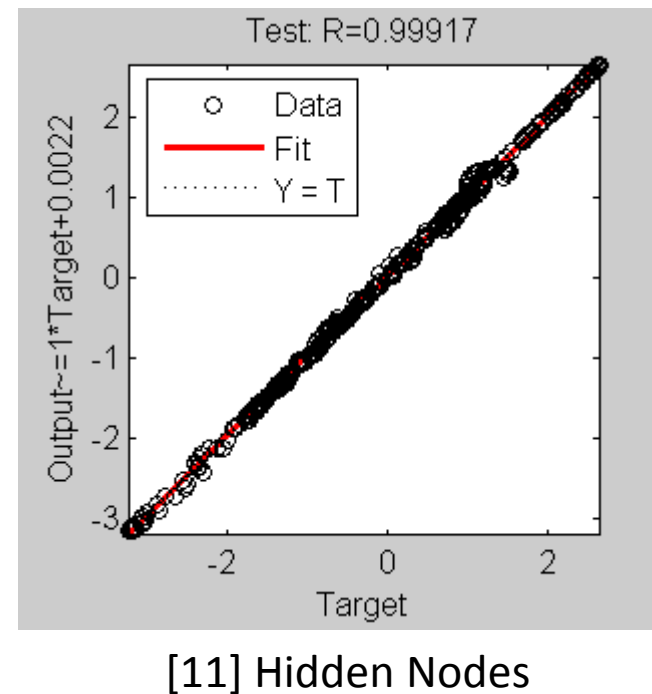
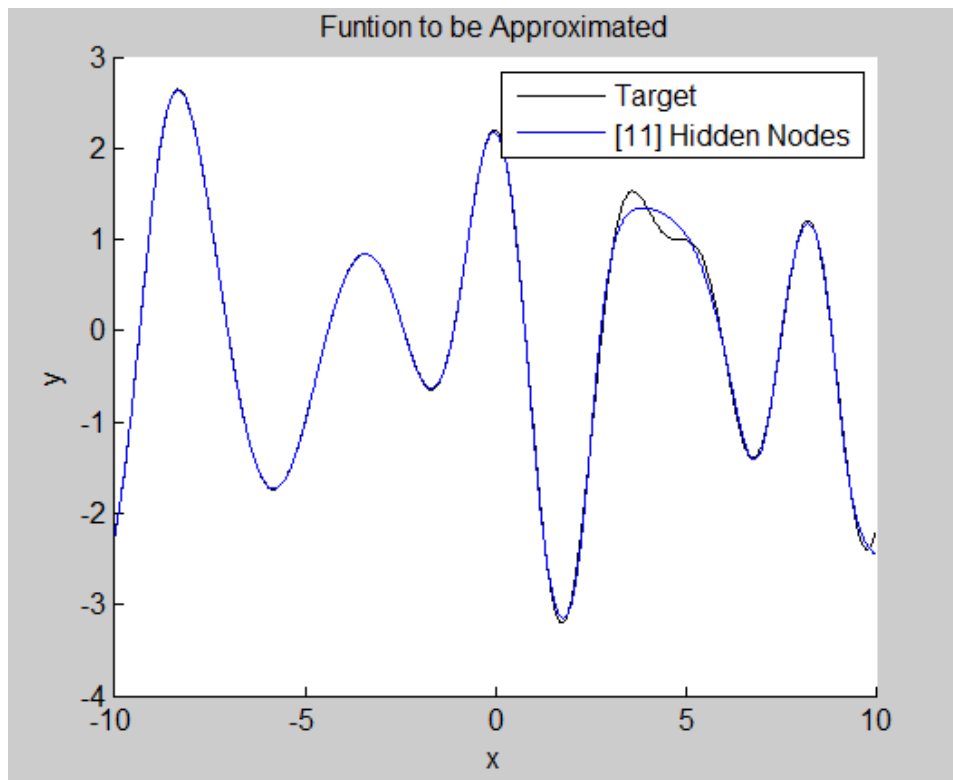


[11] Hidden Nodes

- Not quite as good as [4+4] despite similar number of weights. Why?

Single vs. Multiple Hidden Layers

- Either:
 - Multiple HLs yield more efficient and parsimonious architecture, or
 - Single HLs get more easily stuck in local minima.
- Re-tried several times starting from different initial conditions and found similar performance until, eventually, hit on a better trained network:

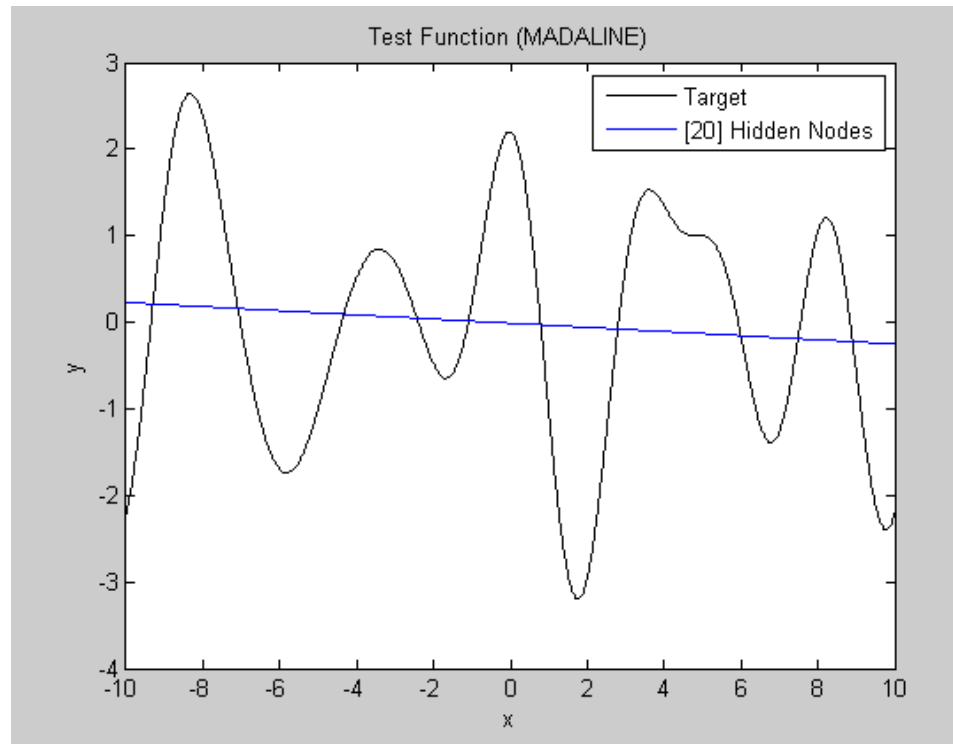


Single vs. Multiple Hidden Layers

- The [4+4] and [11] architectures yielded similar results (with [4+4] being slightly better; also had to try several times with the [11] architecture).
- This offers some empirical evidence that multiple-hidden-layer architectures are equivalent or even slightly better than single-hidden-layer architectures (*after adjusting for the overall number of degrees of freedom*), but that perhaps multiple-hidden-layer architectures have more stable convergence (?).
- A good research project would be to show this empirically for a general class of functions. This would involve doing many runs on artificially-generated data and reporting average performances (a Monte Carlo type study).

What About MADALINEs?

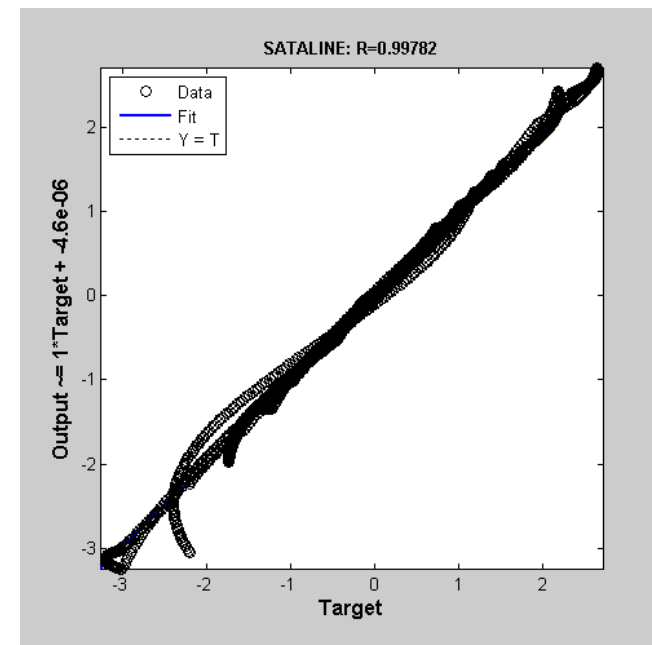
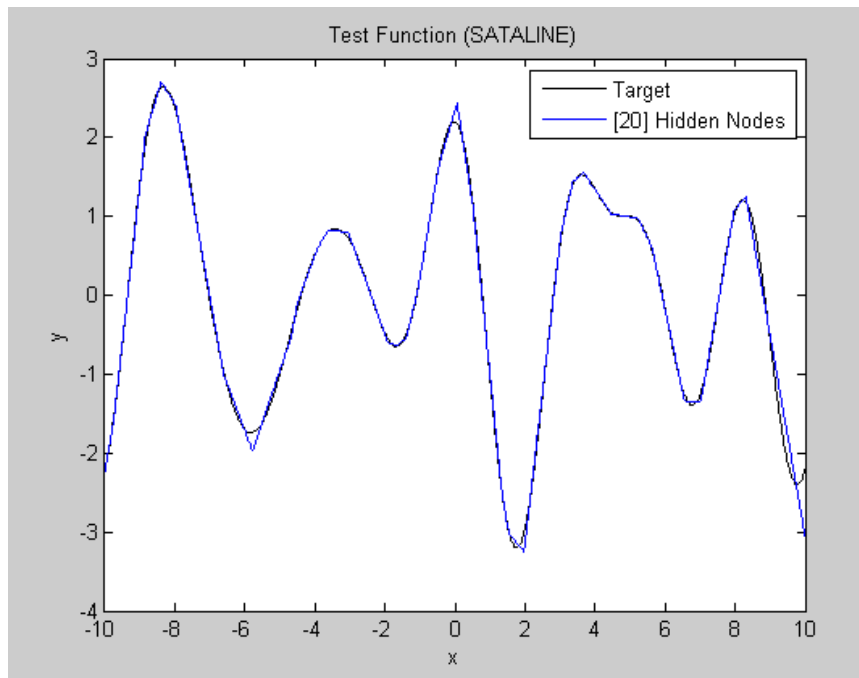
- Can MADALINEs with 20 Hidden Nodes approximate this non-linear function? ...



- Not even close. (Recall that in a HW exercise you showed that, when used as classifiers, MADALINEs can only draw one hyperplane. In some sense, function approximation is akin to a classification problem with many output classes.)

What About SATALINEs?

- Can SATALINEs with 20 Hidden Nodes approximate this non-linear function? ...



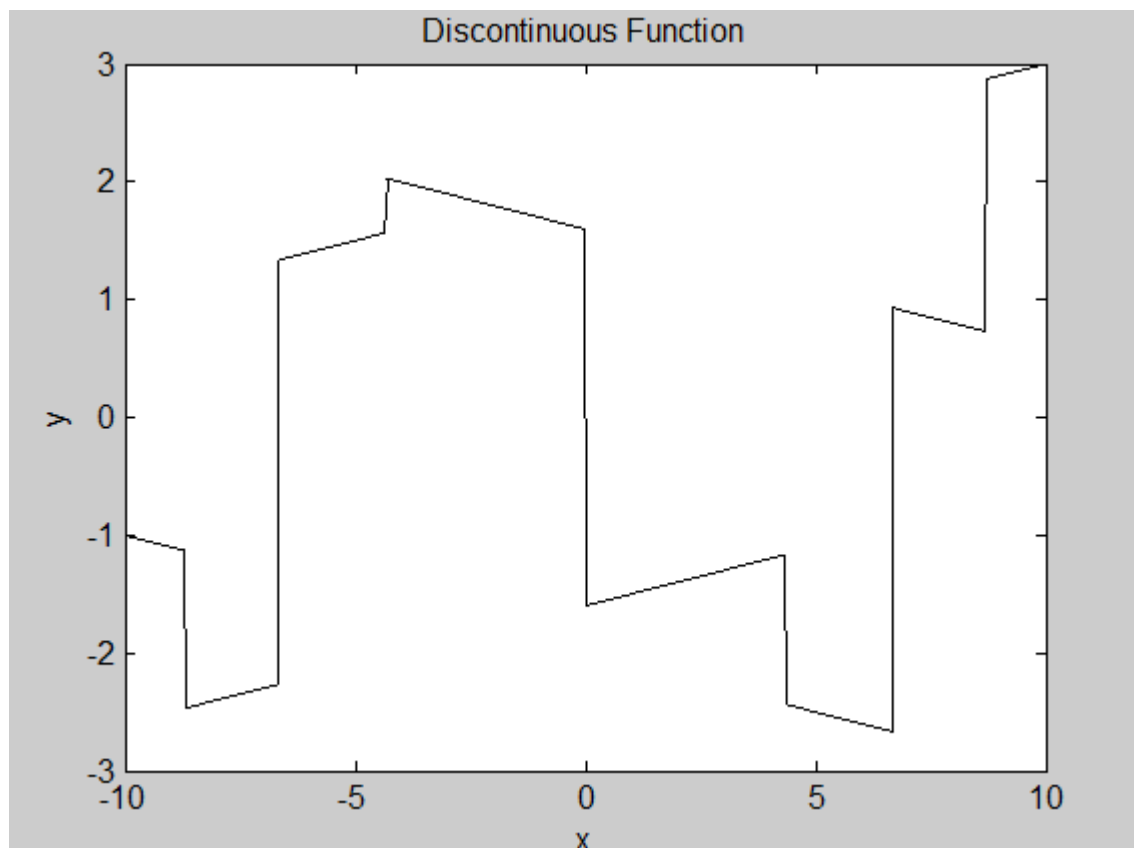
- Yes (albeit a bit jaggedly), because SATALINEs satisfy the non-linear Hidden Nodes requirement for universal approximation.

Aside: Parallelism

- ANNs are inherently parallel and can exploit multicore CPUs and GPUs in both the training and the simulation stages.
- For example, training using Backpropagation over large datasets can use a SIMD approach.
- Many packages now come with parallelization options built in.
- For example, in Matlab, a set of simple flags (e.g. “useParallel \leftarrow yes” and “useGPU \leftarrow yes”) can take care of splitting the data, training, and reassembling the results all “under the hood” transparently to the user.
- Of course, there are also options for micromanaging the process for better load balancing depending on your resources and problem size.
- ...
- Back to Function Approximation...

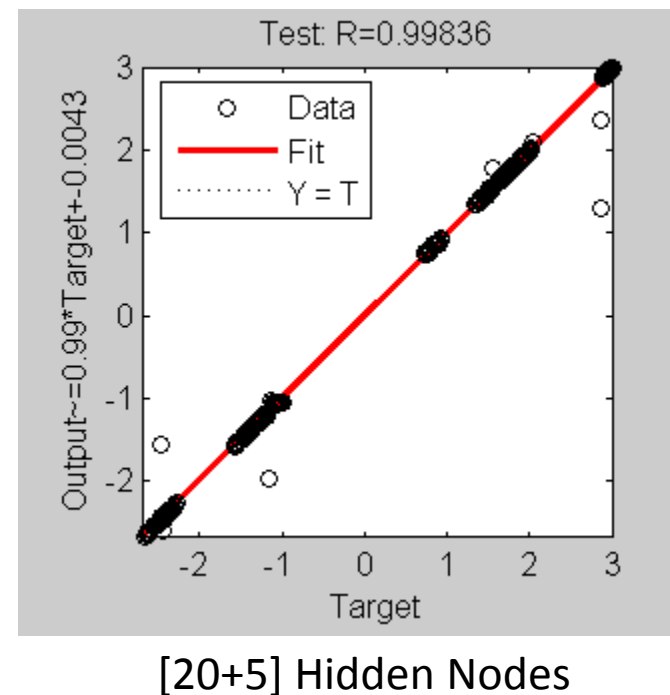
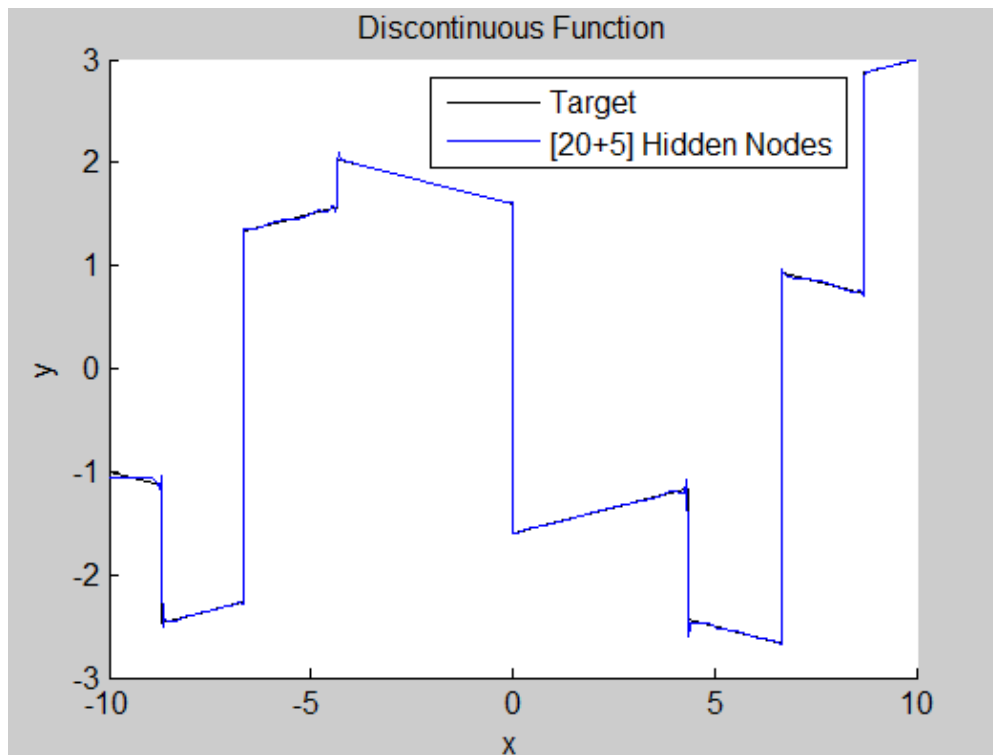
Discontinuous Functions

- FFNNs should be able to approximate discontinuous functions.
- Consider:



Discontinuous Functions

- A [20+5] architecture approximates a discontinuous function very well.
- Other architectures are, of course, also possible.



Noise

- We have seen that any function—highly-nonlinear and even discontinuous ones—can be approximated arbitrarily closely if we have enough hidden-layer nodes (*i.e.*, enough internal parameters or connection weights).
- In real-world applications most target functions are corrupted by a significant amount of noise (particularly when dealing with financial data!).

(Aside: Are Financial Time Series Noisy?)

- Physics Example -- Measure Brightness of a Star
- Measurement includes:
 - Atmospheric aberration
 - Instrument bias/inaccuracies/fluctuations
- When these can't be subtracted or accounted for, we collectively call them "noise."
- Strictly speaking financial price series are "true" observations without extraneous measurement "noise"... Or are they?
- One could argue that price observations do not accurately reflect "fair" prices because of microstructure frictions and uncertainty about future prices (noise?).
- Operationally (and somewhat tautologically), FTS "noise" is whatever we are unable to predict.
- Predictable pattern can be thought of as the "signal"
unpredictable part can be thought of as the "noise."

Back to Noise

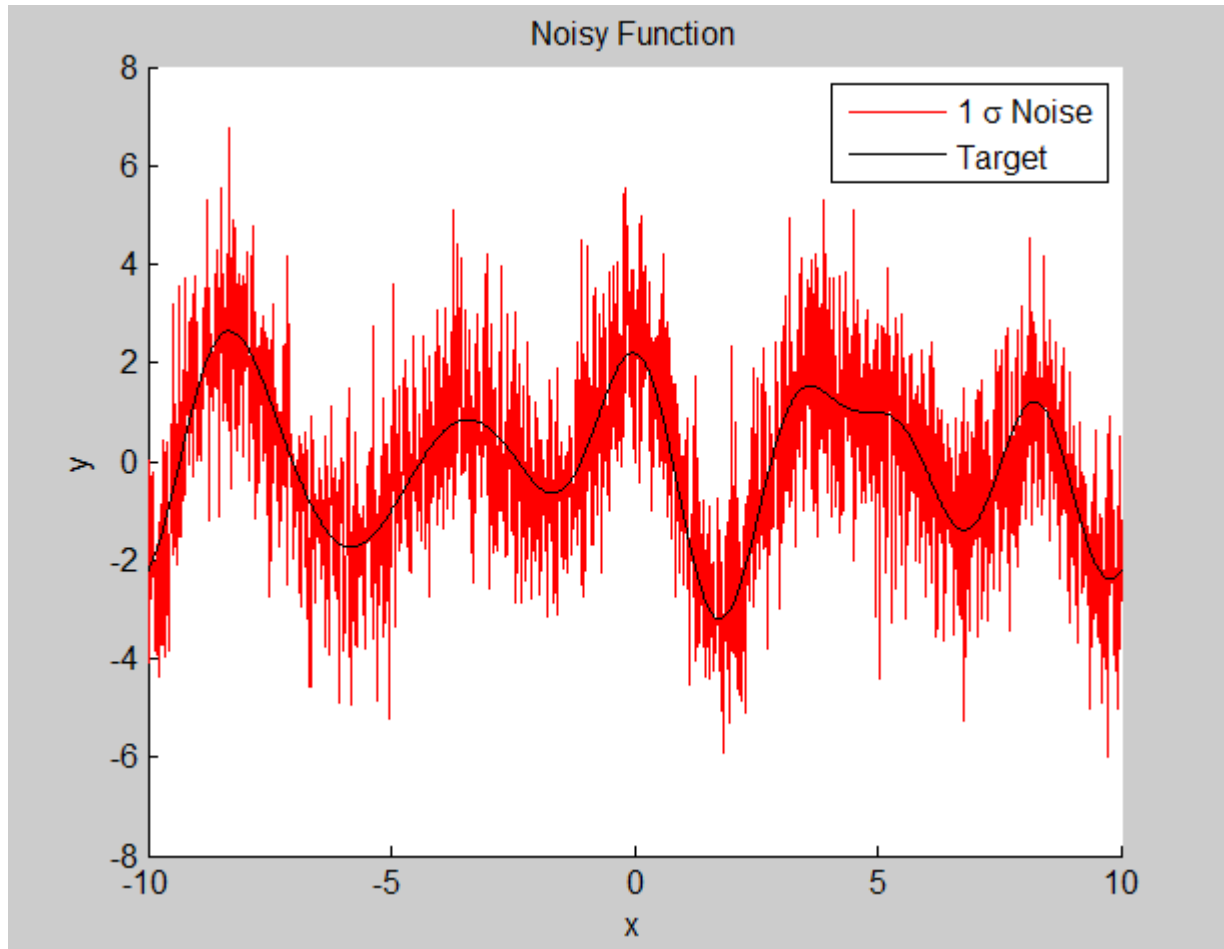
- Consider corrupting our test function with Gaussian Noise:

$$Y_i^* = Y_i + \varepsilon_i,$$

- where $\varepsilon_i \sim \mathcal{N}(0, n\sigma_Y)$, and n is defined as the Noise Strength
- while $1/n$ is defined as the the Signal-to-Noise ratio.

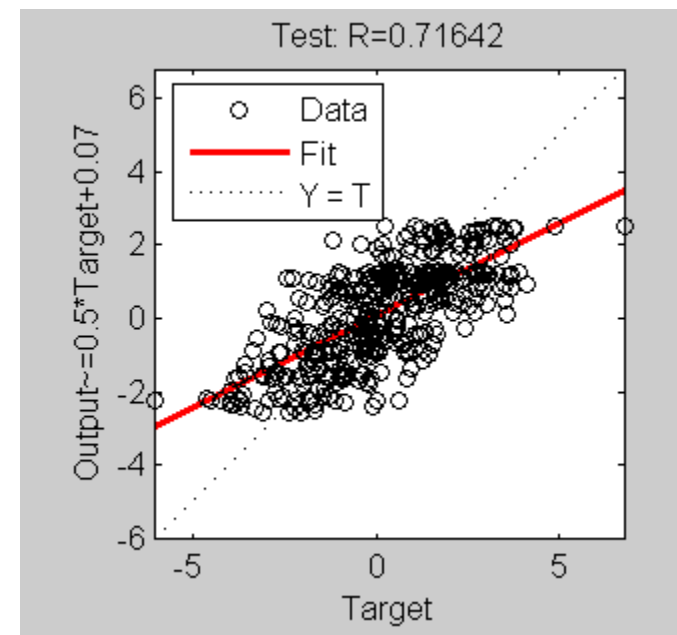
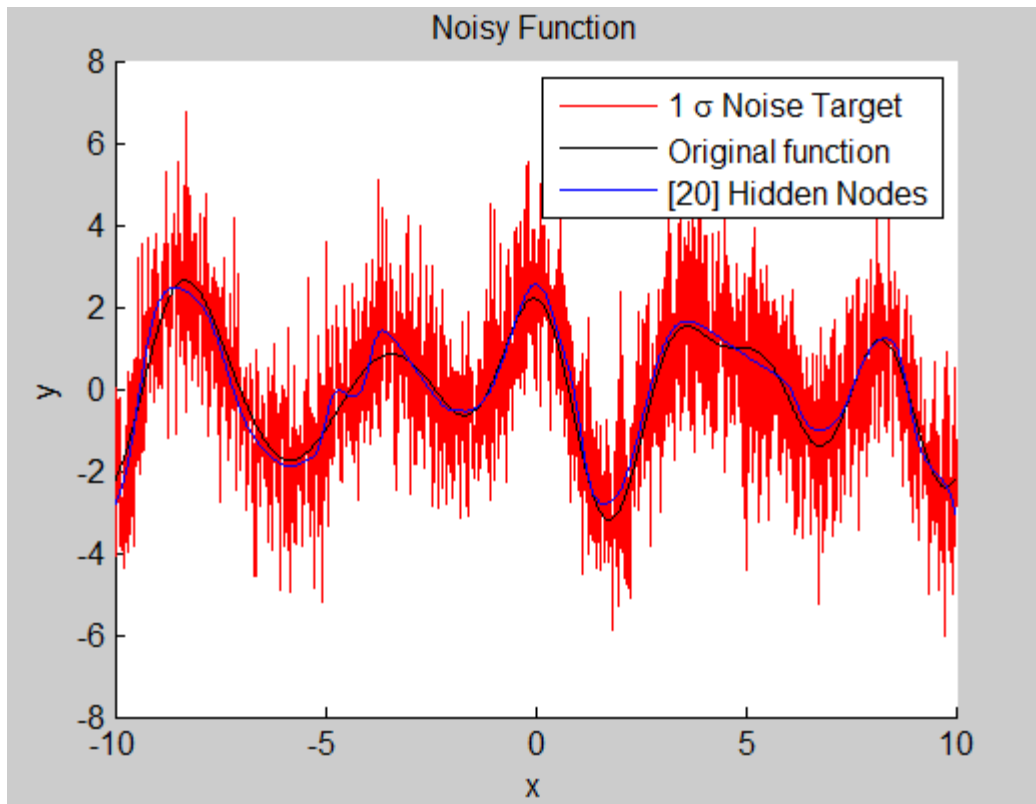
Noise

- Noise Strength of 1 Standard Dev:



Noise

- A [20] FFNN fitted on the 1 Std Dev Noisy Data (red) approximates the original function (black) quite well.
- Regression curve is now more scattered, but has reached the theoretically maximum performance...



[20] Hidden Nodes

Theoretically Maximum Performance

- We saw before that for non-noisy targets the R^2 can reach 1.
- For noisy targets the situation changes and the R^2 will be less than 1.
- We can, in fact, derive an expression for the theoretically maximum R^2 for a given level of noise.

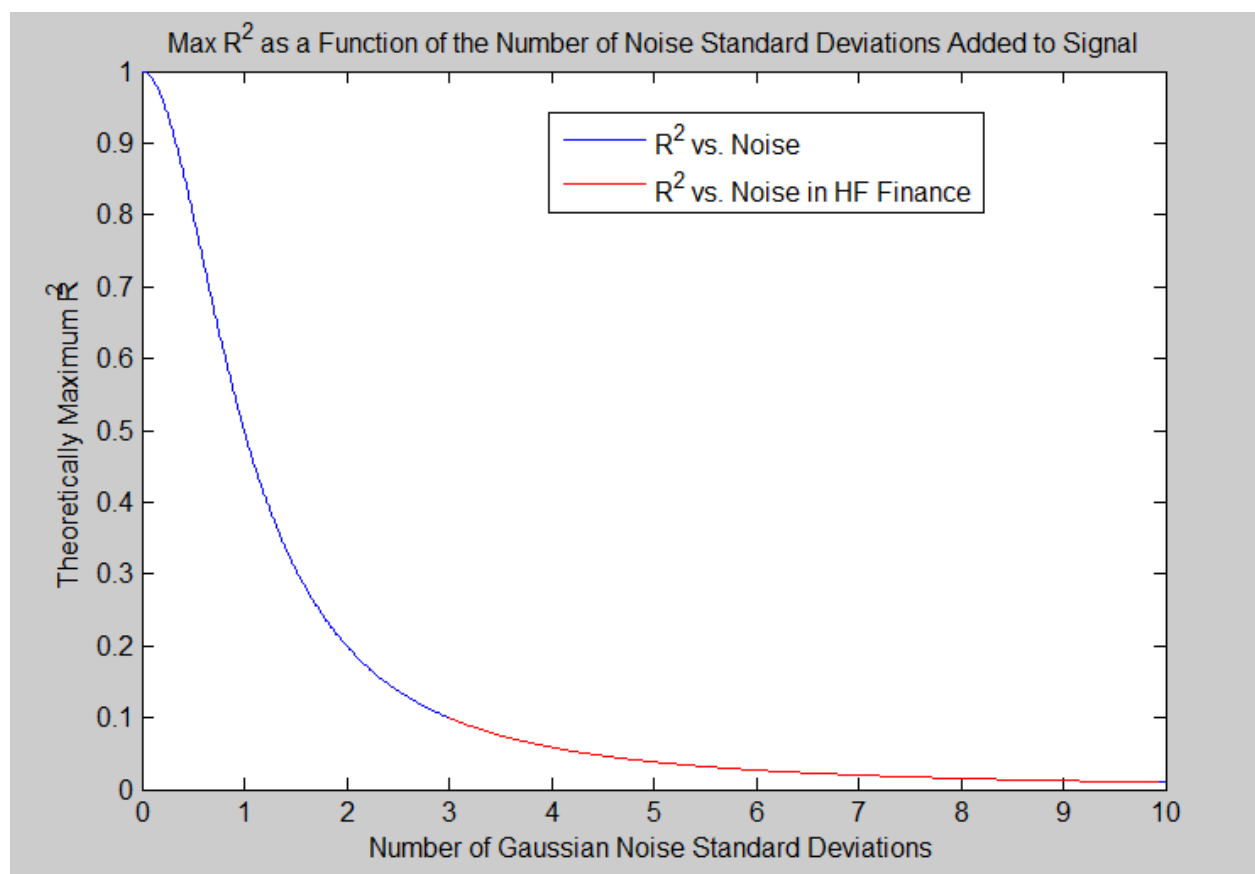
$$\max R^2 = 1 - \frac{\sum_i (Y_i^* - Y_i)^2}{\sum_i (Y_i^* - \bar{Y}^*)_i^2},$$

from which it follows that $\max R^2 = 1 - \frac{n^2}{1+n^2}$.

- For $n=1$, the maximum R^2 is 0.5, which is precisely what was reached by the [20] network above ($R=.71$).

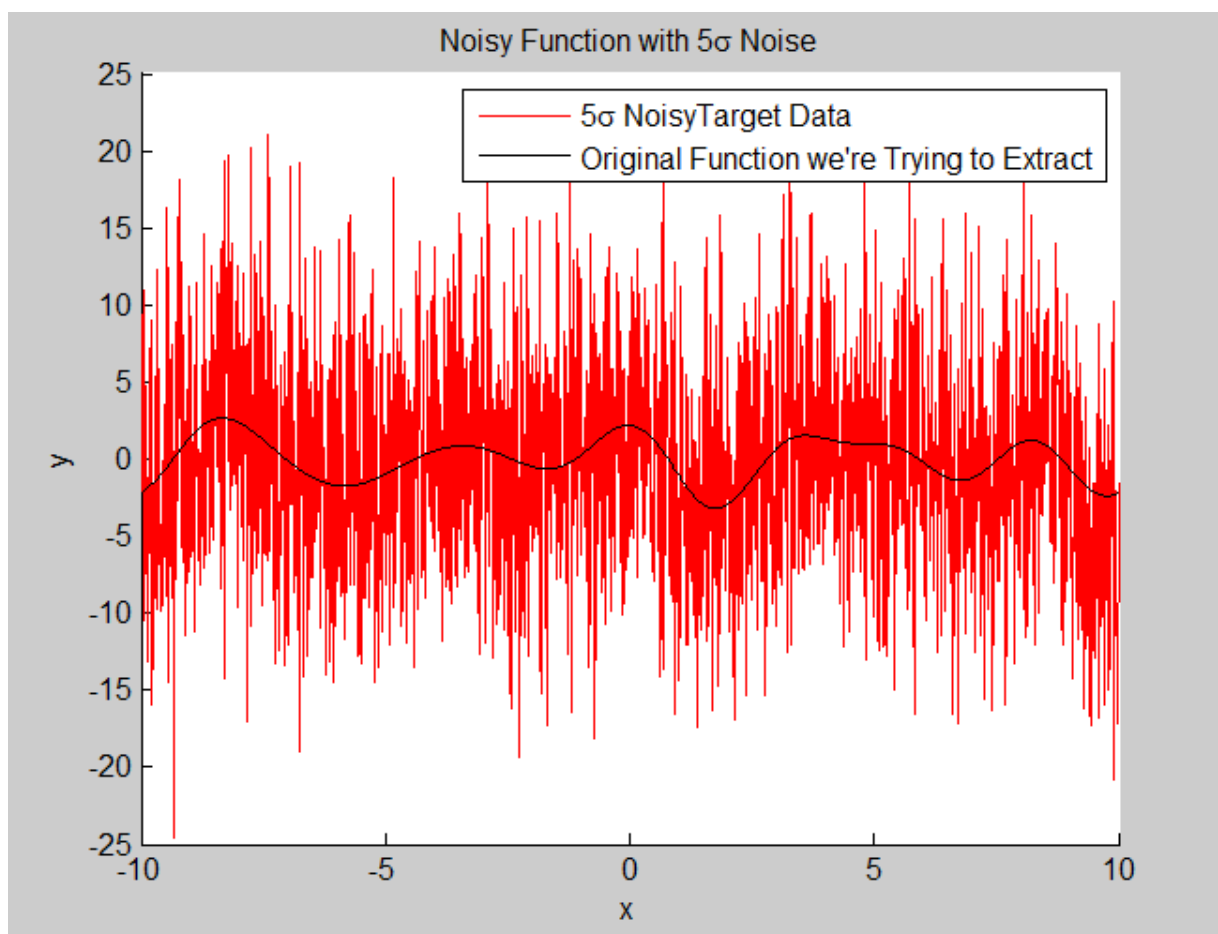
Theoretically Maximum Performance

- In HF Finance R^2 s typically hover in the single digits.
- This means that, assuming our models capture most of the R^2 , the signal-to-noise ratios ($1/n$) are around 0.1-0.3 .



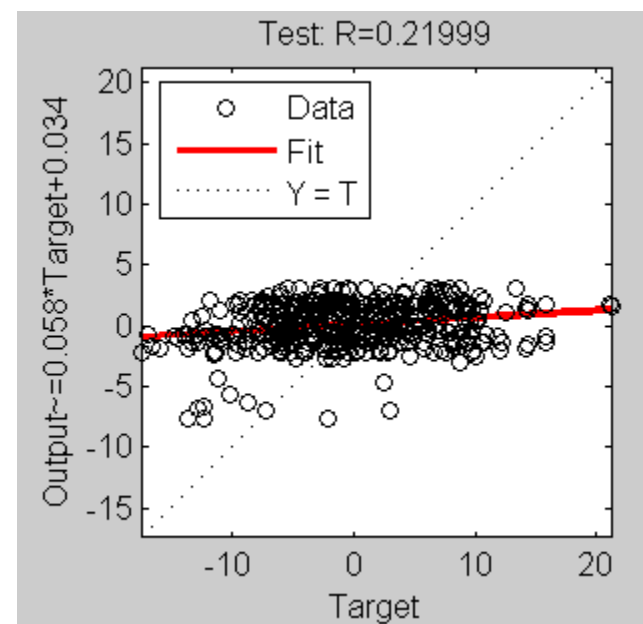
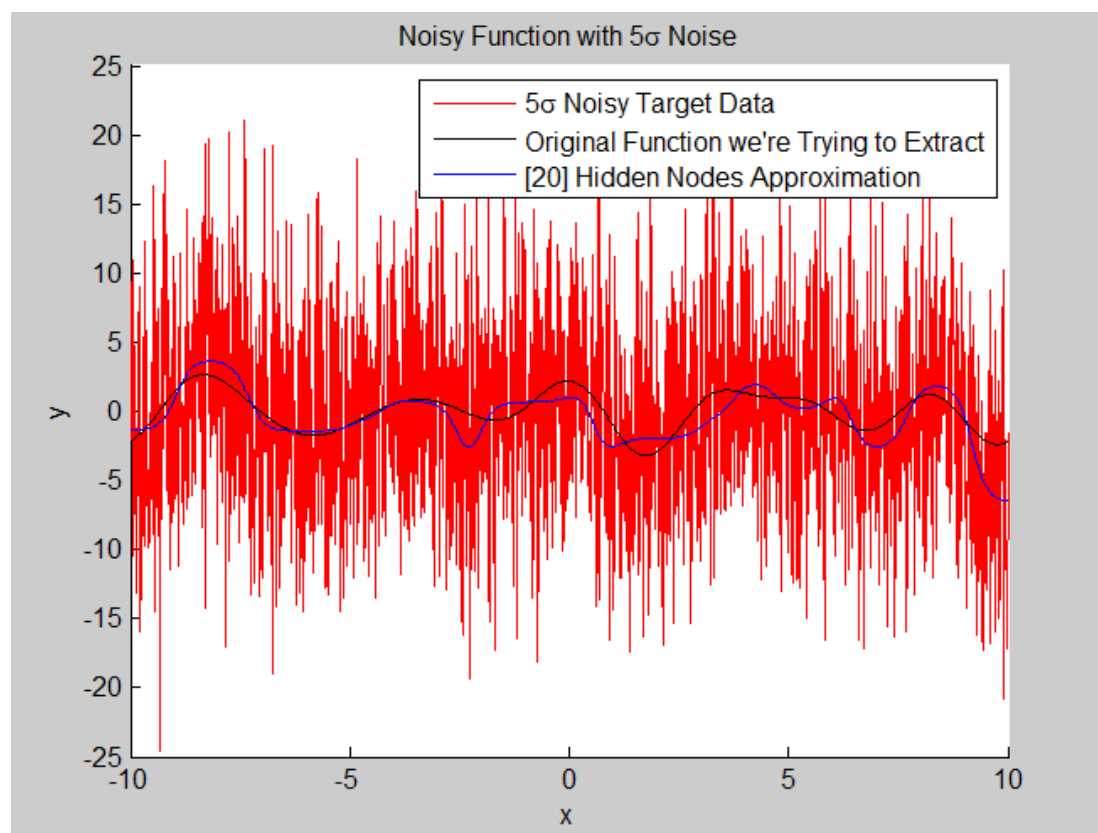
Noise

- A more realistic noise strength is $n=5$ (5σ).
- Theoretically Maximum $R^2 = 0.039$.



Noise

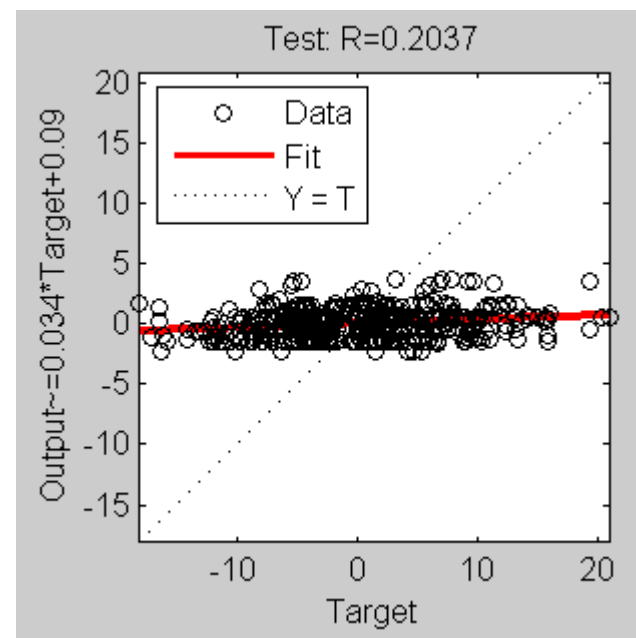
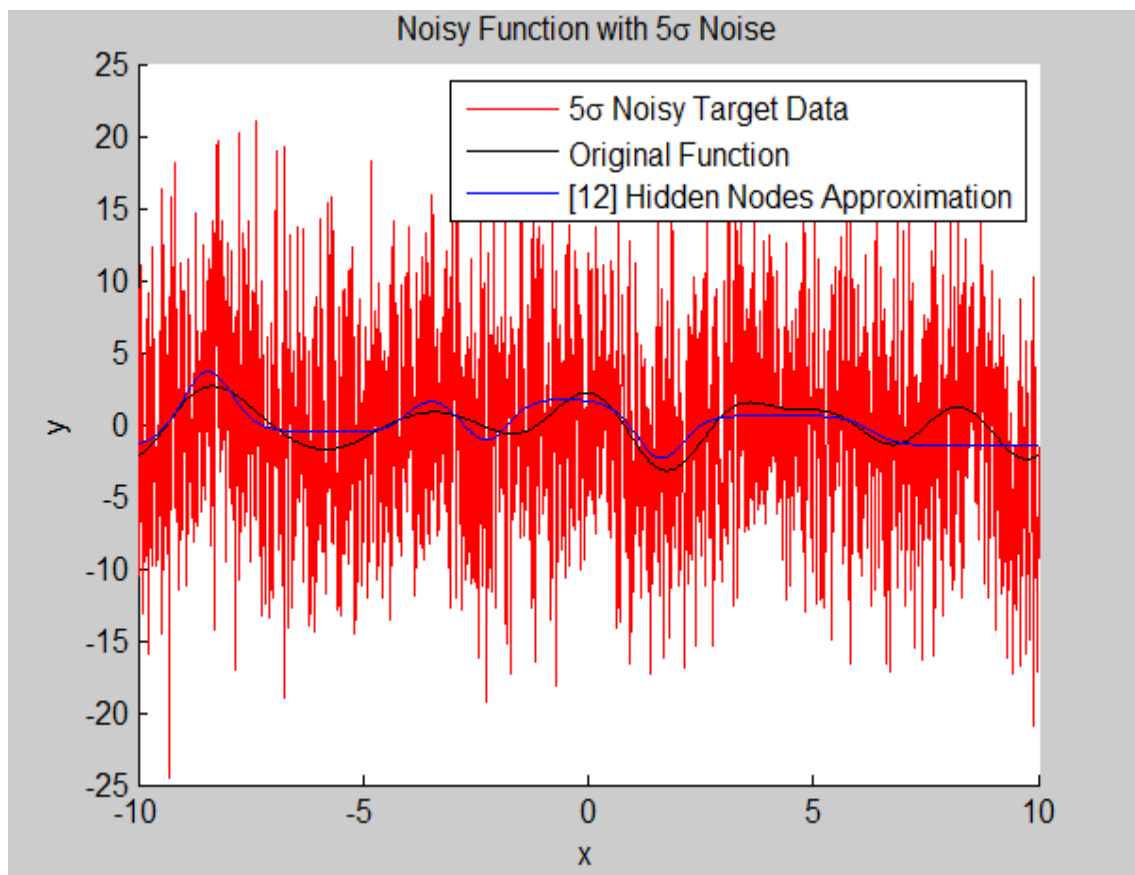
- Noise strength is $n=5$ (5σ).
- Theoretically Maximum $R^2 = 0.039$ ($\max R = 0.20$).
- Slight overfit with [20] Hidden Nodes, or artifact of data size?



[20] Hidden Nodes

Noise

- Noise strength is $n=5$ (5σ).
- Theoretically Maximum $R^2 = 0.039$ ($\max R = 0.20$).
- [12] Hidden Nodes may be slightly better:



[12] Hidden Nodes

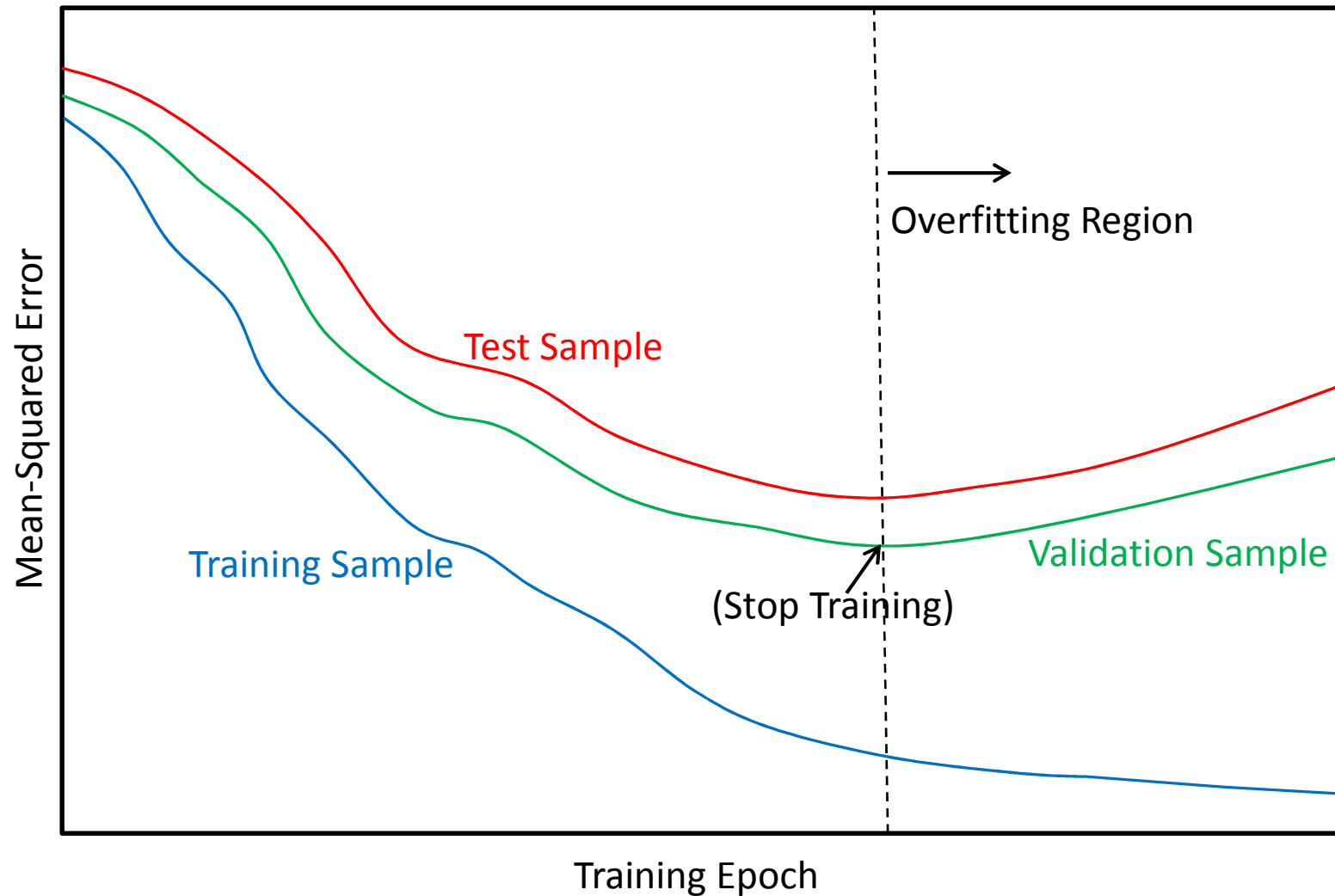
Overfitting

- Compounded by too many degrees of freedom (parameters, weights), too few data points.
- Leads to poor Generalization, i.e., the model's ability to perform well when presented with data that it has never "seen" before (i.e., data that was not used during the fitting or training of the model).
- Balancing act: The model should be as complex as needed, but no more (Parsimony).
- Remember, we don't have the original function (black curve above) available to us.
- One way to get around overfitting is to start with a relatively large model (number of hidden nodes), split the data into Training and Validation samples, and use the validation sample to stop the training. This is the method of Early Stopping or Cross Validation.
- Model performance stats such as R^2 are then reported on yet another data sample called the Testing or Hold-Out sample, which was not used either for training or for stopping training.
- Note that Validation and Testing are often used interchangeably in nomenclature, but they are different, and we must be clear about which definition is being used.

(Aside: Data Mining)...

- In fields outside of Finance “*Data Mining*” is a perfectly respectable term referring to the analysis and extraction of information out of (typically large) corpuses of data.
- In Finance (particularly HFT), Data Mining is a “dirty word”, and it refers to finding patterns that do not persist in the data (unlike in other fields such as bioinformatics, physics, marketing, etc.).
- This is likely due to the small predictable signals present in market data since any predictable patterns are usually arbitrated away quickly, and one ends up fitting to noise.
- Typically one needs a great deal of data to make robust predictions, but there is a tradeoff between amount of data and how far back to go in the data (staleness) to dig for patterns.
- For example, in a random Head/Tail data set: HTHHTHHT one may be tempted to find a pattern such as “Heads are more likely after a Tail,” which is an artifact of the smallness of the sample: (“Data Mining” as used in Finance).

Early Stopping



- Stop Training when Validation Error reaches a minimum.
- For a large model, Training Error usually continues to decrease past this point.

Regularization

- Early Stopping doesn't directly modify the FFNN architecture.
- Early Stopping doesn't use all data for training.
- Early Stopping requires more data.
- Regularization: Avoids overfitting by penalizing complexity.
- Pruning, Penalty for Complexity, Bayesian Methods.

Regularization: Pruning

- Remove weights that are “small”:
 - If $|w_{ij}| < \delta$ then set $w_{ij} \triangleq 0$;
 - Retrain Network;
 - Repeat until no more “small” weights.
- “Jiggle” weights and remove those that have “small” impact on the output (or error)—
Sensitivity:
 - If $\left| \frac{\partial \text{output}}{\partial w_{ij}} \right| < \delta$ then set $w_{ij} \triangleq 0$;
 - Retrain Network
 - Repeat until no more “small” Sensitivities left.

Regularization: Pruning

- Pruning Regularization Methods work reasonably well at producing parsimonious models.
- By setting weights to zero, they adjust the Network's architecture.
- We can re-train with fewer nodes to reflect the number of weights (parameters) left after pruning.
- Can be used in combination with Early Stopping (if have enough data).
- Drawbacks:
 - Reliance on arbitrary “small” parameter δ .
 - Computationally intensive/Time consuming.

Regularization: Penalty

- Penalty function so far has been MSE:

$$\mathcal{F}_W = E = \frac{1}{N} \sum_i \frac{1}{2} (t_i - a_i)^2.$$

- Introduce a term to directly penalize complexity by penalizing weight magnitudes:

$$\mathcal{F}_W = (1 - \lambda) \frac{1}{N} \sum_i \frac{1}{2} (t_i - a_i)^2 + \lambda \sum_{ij} \frac{1}{2} w_{ij}^2.$$

- Notice we're keeping smoothness (differentiability) criterion for automated training.

Regularization: Penalty

- Penalty Regularization works reasonably well.
- Can be used with the other methods discussed above (Early Stopping and Pruning).
- Main drawback is its reliance on *ad-hoc* weight-complexity penalty “knob” λ .

Annealing Methods

- Start with a “large” architecture.
- From a set of weights (“*state*” of the system) in the current iteration, $\mathbf{w}(m)$, obtain new weights $\mathbf{w}(m + 1)$ (the new state) by some preferred method (either with or without regularization, as above).
- Compute the new penalty function (the “*Free Energy*” which we’re trying to minimize): $\mathcal{F}_W(m + 1)$.
- Transition to the new state (*i.e.*, accept the new weights) with probability (the Boltzmann Probability):

$$P \propto \exp \left\{ -\frac{\mathcal{F}_W(m + 1) - \mathcal{F}_W(m)}{T(m)} \right\},$$

where $T(m)$ is the “*Temperature*” which is decayed slowly.

Annealing Methods (cont.)

- Do not transition to the new state (*i.e.*, reject the new weights) with probability $1 - P$, and corrupt the previous weights $\mathbf{w}(m)$ with a small amount of noise dependent on the current Temperature:

$$\mathbf{w}(m) \leftarrow \mathbf{w}(m) + \varepsilon(0, \sigma(T(m))).$$

- Decay the Temperature (*i.e.*, the Noise Variance) slowly only if transition to the new state.
- Repeat until Temperature or Penalty is “small”.
- Inspired by metallurgical or ceramic annealing process (slow cooling to form stable alloys/crystals).
- A close cousin to Genetic Algorithms (another variant used with NNs)
- Process converges *in probability* to global optimum.
- Works well, but:
 - It is slow and computationally very intensive.
 - Works only for small to medium size problems.
 - Depends on a bunch of *ad hoc* parameters.

Bayesian Methods

- Bayesian methods can be used for regularization.
- They have a number of advantages such as less reliance on ad-hoc parameters and other heuristics, better use of data samples, objective criteria for comparison among models, can help improve architecture construction.
- We'll take a detour to consider Bayesian methods and come back to Neural Networks later.

Bayesian Methods

- Thomas Bayes addressed the problem of inference with his famous theorem published posthumously in 1763.
- The modern form of Bayes' Theorem (BT) was rediscovered by Laplace in 1812. Laplace framed it with far more clarity than Bayes, and used it for solving problems in celestial mechanics, medical statistics, and even jurisprudence.
- BT fell out of favor despite Laplace's successes, because it was considered too subjective and not rigorous enough (wrongly!).
- Frequentist definition of probability was considered more objective/rigorous: based on long-run relative frequency of events given infinite trials.
- However, frequentist definition suffers from practical problems (infinite trials, etc.).

Bayesian Methods

- For example, Laplace computed the posterior pdf for the mass of Saturn, M , given orbital data from astronomical observations:

“It is a bet of 11,000 to 1 that the error of this result, M , is not $1/100^{\text{th}}$ of its value.”

- He would’ve won the bet, since he was within 0.63% of the today’s best estimate.
- Note that: The mass of Saturn is constant and not a random variable, and thus has no frequency distribution for the frequentist method to draw from.
- Frequentist’s “randomness” and Bayesian’s “uncertainty” are equivalent, and both are “rigorous.”
- One can make a good argument that all inductive or empirical claims (including historical claims, such as “Caesar crossed the Rubicon”) can be framed in terms of BT.

Bayesian Methods

- To illustrate, consider a bag with 7 red marbles and 5 blue marbles. The bag is shaken and a marble selected “at random.” For the 1st draw we have:

$$P(\text{red}) = 7/12, \text{ and } P(\text{blue}) = 5/12.$$

- If we remove another marble, the probabilities will depend on the outcome of the 1st draw.
- Problem: Suppose we are not told the outcome of the 1st draw, but are told the outcome of the 2nd draw. Do the probabilities of the 1st draw change?
- Most of us would say no since the 2nd draw doesn’t affect the first draw.
- But consider the extreme example where we have only one red and one blue marble.
- The outcome of the 2nd draw completely changes the probabilities of the 1st draw.
- The 2nd draw does not affect the 1st draw physically, but knowledge of the 2nd draw affects knowledge of the 1st draw.
- Conditional probabilities represent logical connections that could be missed by a strict frequentist approach.
- A similar problem is the “Monty Hall Problem” which even experts tend to get wrong on the first pass.
- These are naturally solved using BT.

Bayesian Methods

- Bayes' Theorem (BT):

$$P(X|Y, I) = \frac{P(Y|X, I) \times P(X|I)}{P(Y|I)},$$

where X and Y are RVs and I is background information.

- The following is a useful BT expression:

$$P(h|data, I) \propto P(data|h, I) \times P(h|I),$$

Where h is a “hypothesis,” and the proportionality constant is fixed by normalization.

- The idea is to relate the probability of interest, $P(h|data, I)$ (the *Posterior Probability*), to probabilities that are easier to obtain.

Bayesian Methods

- Anatomy of BT:

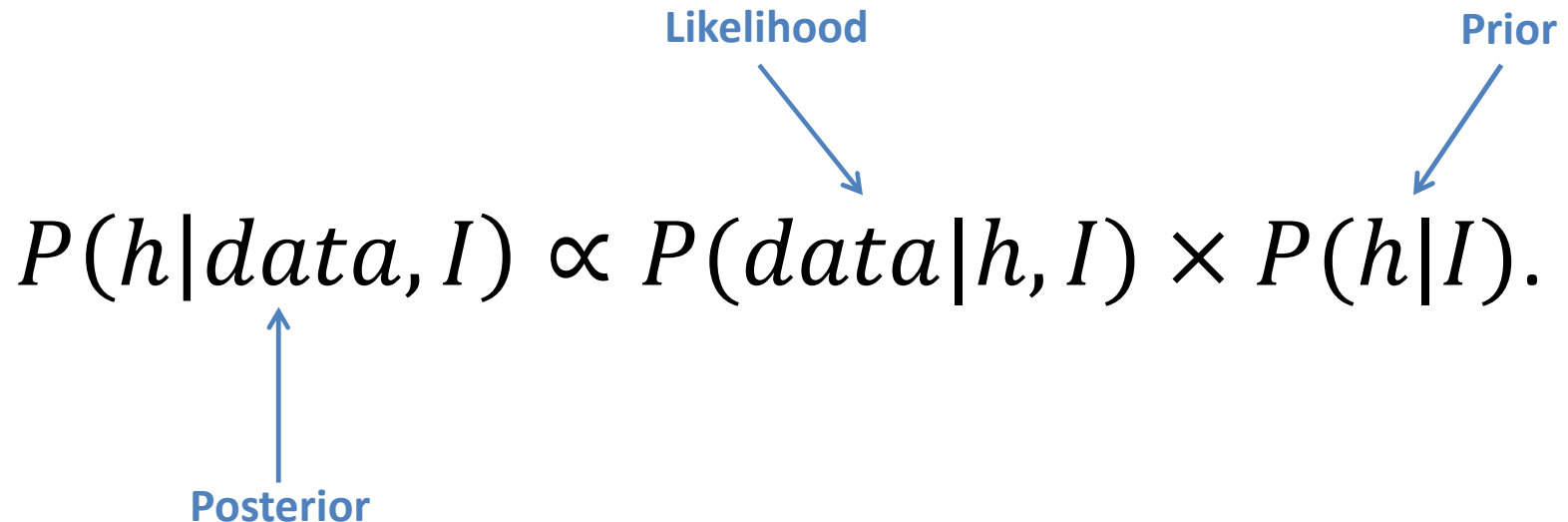
The diagram shows the Bayesian formula with four labels and arrows indicating their components:

- Likelihood**: An arrow points from this label to the term $P(data|h,I)$ in the numerator.
- Prior**: An arrow points from this label to the term $P(h|I)$ in the numerator.
- Posterior**: An arrow points from this label to the term $P(h|data, I)$ on the left side of the equation.
- Evidence**: An arrow points from this label to the term $P(data|I)$ in the denominator.

$$P(h|data, I) = \frac{P(data|h,I) \times P(h|I)}{P(data|I)}.$$

Bayesian Methods

- For Parameter Estimation, the Unnormalized form of BT is usually good enough:


$$P(h|data, I) \propto P(data|h, I) \times P(h|I).$$

The diagram illustrates the components of the Bayesian theorem equation. Three blue arrows point to the terms in the equation: one from 'Likelihood' to $P(data|h, I)$, one from 'Prior' to $P(h|I)$, and one from 'Posterior' to $P(h|data, I)$.

Bayesian Methods

- Example: Is a coin fair? Let the coin bias towards obtaining “Heads” be specified by H , such that if the coin is fair, we assign a high belief to H being around 0.5.
- We carry out an experiment of coin flips, and we measure $P(H|data, I)$.
- The Prior state of knowledge with least information (most ignorance) is a uniform distribution:

$$P(H|I = \emptyset) = \begin{cases} 1 & 0 \leq H \leq 1, \\ 0 & \text{otherwise.} \end{cases}$$

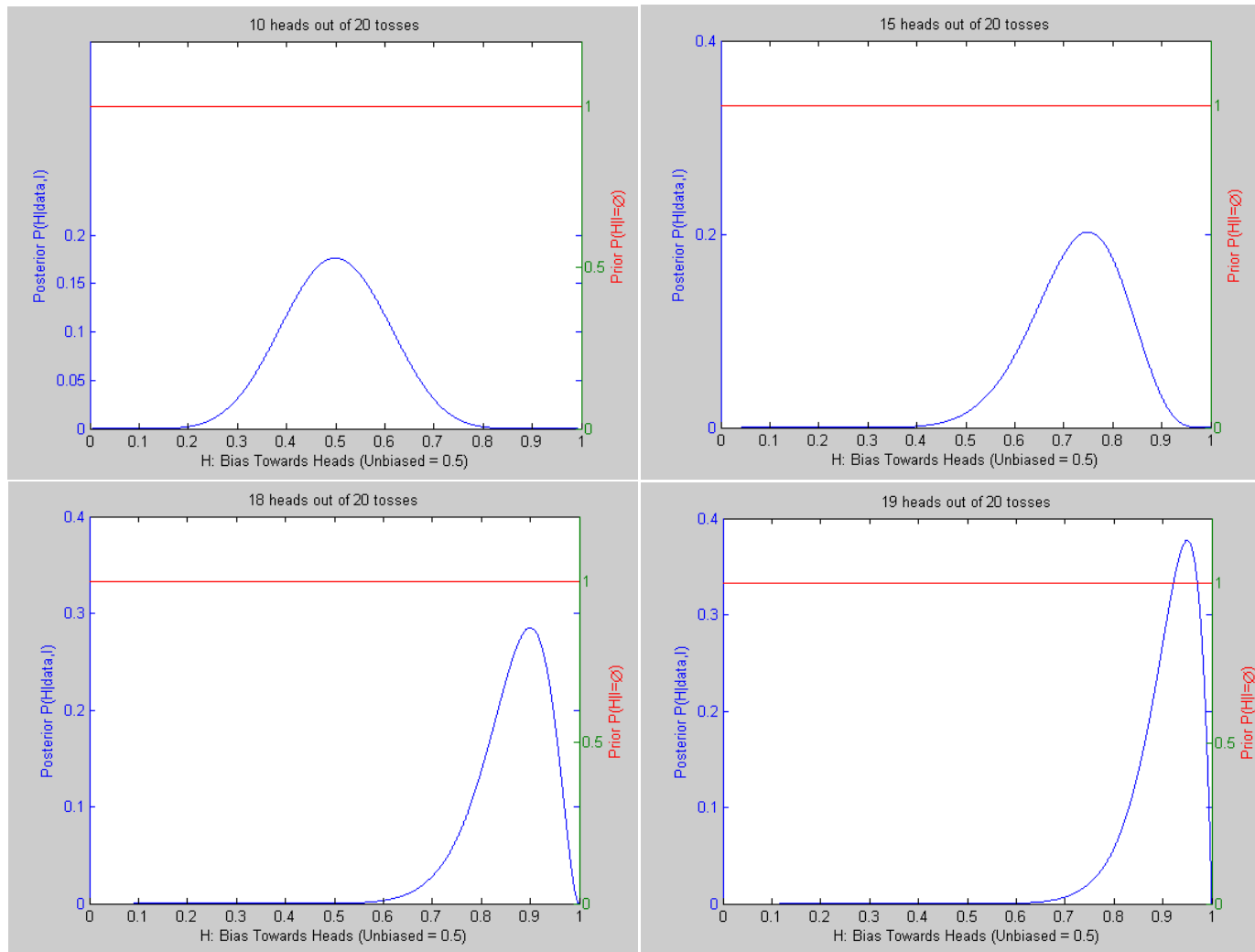
- The prior is modified by knowledge acquired from data through the Likelihood:

$$P(data|H, I) \propto H^R (1 - H)^{N-R},$$

which is a binomial with R tails out of N tosses.

- From BT, the Posterior $P(H|data, I)$ is proportional to the product of these two distributions.
- The Bayesian Posterior encodes our inference about the value of a parameter (H) given the relevant data (coin tosses) and any prior information.

Bayesian Methods



- Given observed data (coin tosses) the Bayesian approach allows us to assign a likelihood to the value of the parameter H (the Posterior Probability).

Bayesian Methods

- Model Selection Problem: Model A and Model B describe data D , but model B has an adjustable parameter b . Which one is better?
- Consider the Posterior Ratio $Q = \frac{P(A|D,I)}{P(B|D,I)}$, which is much larger than 1 if A is much more likely than B , and close to zero if vice-versa. BT gives:

$$Q = \frac{P(D|A,I)}{P(D|B,I)} \times \frac{P(A|I)}{P(B|I)}.$$

- We can set the prior ratio to 1 (assuming no prior preference between models), and deal with the nuisance parameter b :

$$P(D|B,I) = \int P(D, b|B, I)db = \int P(D|b, B, I)P(b|B, I)db,$$

where the second integral contains an ordinary likelihood and prior.

Bayesian Methods

- Assume that b is in some range b_{min} to b_{max} , so we can assign a naïve uniform prior $P(b|B, I)$, and assume that there is an optimal b^* which yields closest agreement with D (otherwise the model would be of questionable value), with some dispersion δb (to reflect any noise in D not captured by B). We can then show that:

$$P(D|B, I) \cong P(D|B^*, I) \times \frac{\delta b}{b_{max} - b_{min}},$$

where $P(D|B^*, I) = P(D|b^*, B, I)$ is the likelihood that D is explained by the optimal model B (i.e., the B that uses b^* , namely the optimal b).

- We thus have, for the Posterior Ratio:

$$Q \cong \frac{P(D|A, I)}{P(D|B^*, I)} \times \frac{b_{max} - b_{min}}{\delta b}.$$

- The first term weighs the relative performance of the two models, adjusted by the second term (the Occam Factor), which penalizes B for the adjustable parameter b .
- “Goodness of Fit” cannot be the only factor in choosing models, as this would always favor B . The Occam Factor takes care of this problem.

Bayesian Methods

- If both A and B have adjustable parameters one can show that the Posterior Ratio is:

$$Q \cong \frac{P(D|A^*, I)}{P(D|B^*, I)} \times \frac{\delta a(b_{max} - b_{min})}{\delta b(a_{max} - a_{min})},$$

where a is A 's adjustable parameter.

- We can also think of A and B as hypotheses to be compared, etc.
- Thus, the Bayesian Method is powerful and can allow us to estimate parameters (as in the Coin-Toss example), as well as for selecting the most powerful and parsimonious model.
- Bayesian Methods can thus be applied to regularization by limiting the number of parameters.
- This is called “Bayesian Regularization.”
- Back to Neural Networks...

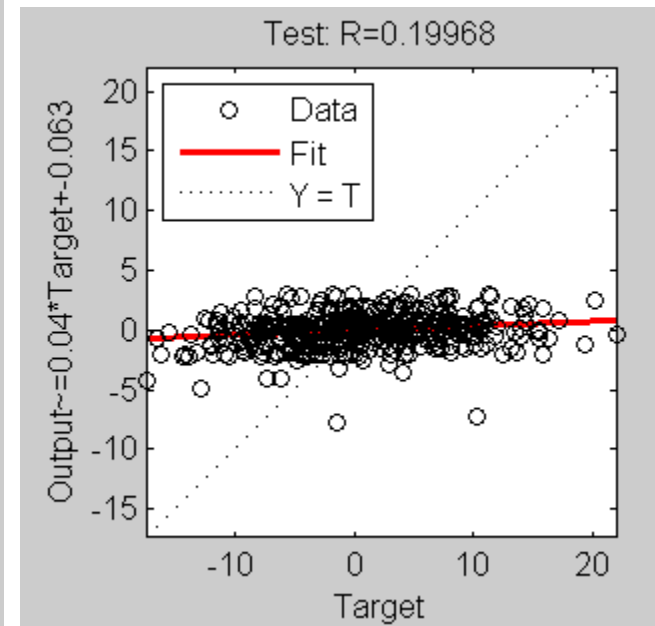
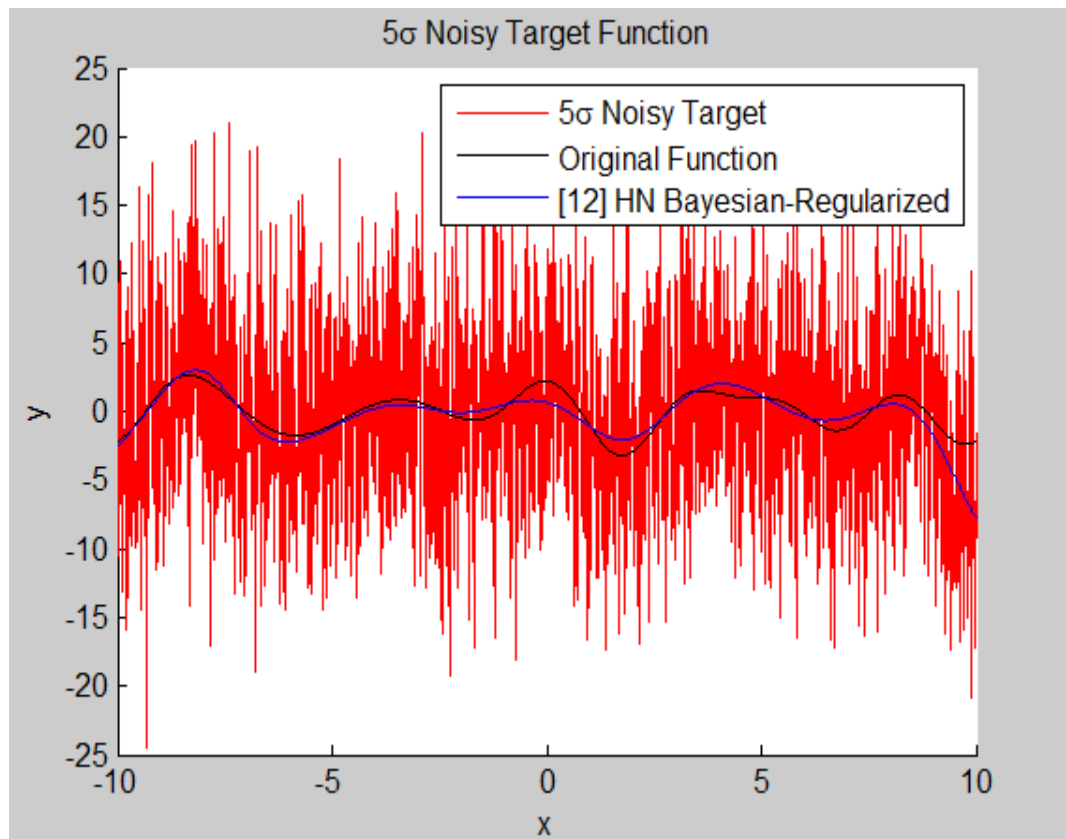
Bayesian Regularization

- The previous Regularization and Early-Stopping methods have either a host of *ad hoc* parameters needing heuristics to specify them (*i.e.*, lacking objective methods for specifying them), or diminish the overall data size by setting aside Test and validation samples, or are computationally intensive, or all of the above.
- Bayesian Regularization is a method that (*):
 - Introduces objective criteria for regularizing parameters;
 - Introduces objective criteria for comparing among models (including non-neural network approaches);
 - Does not decrease the data size;
 - Bonus: Obtains the Effective Number of Degrees of Freedom or Internal Parameters (weights).
 - This can be used to re-train a new network with a smaller architecture that matches the Effective Number of Weights found, as above, or to check if a larger network leads to the same Effective Number of Weights, meaning it's unnecessary to go larger.

(*) Ref: MacKay, *Neural Computation*, Vol. 4, No. 3, 1992, pp. 415–447

Bayesian Methods

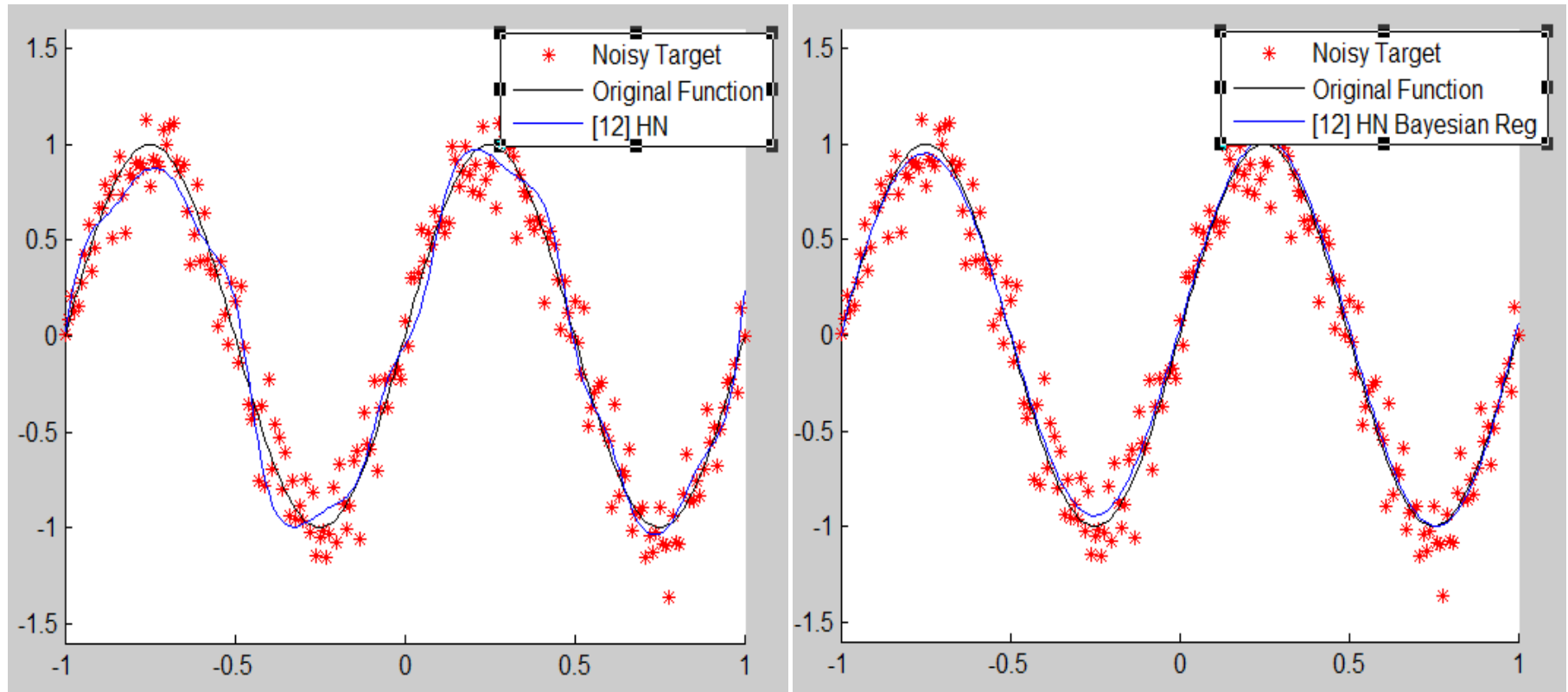
- Bayesian Regularization produces smoother fit:



[12] HN, Bayesian-Regularized

Bayesian Methods

- Bayesian Regularization produces smoother fit:



- Same no. HNs to start, but the Number of Effective Parameters went from 37 to 9.
- R^2 's comparable (around the max), though a bit higher for BR.
- But fit is smoother with Bayesian Regularization (better Generalization).

Case Study: House Values in Boston

- In Suffolk County, Massachusetts, property tax values are assessed relative to the previous year's *median* house market value.
- You are a financial consultant and the Suffolk County Treasurer and Tax Collector has requested your help with the following problem:
 1. There has been a redistricting of several Boston suburbs.
 2. The treasurer wants a good estimate of the median prices in the new suburbs, but he lacks recent prices for most of the homes in the new suburbs.
 3. He provides you with the following data on Boston neighborhoods(*):

Data Set Information:

Concerns housing values in suburbs of Boston.

Attribute Information:

1. CRIM: per capita crime rate by town
2. ZN: proportion of residential land zoned for lots over 25,000 sq.ft.
3. INDUS: proportion of non-retail business acres per town
4. CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
5. NOX: nitric oxides concentration (parts per 10 million)
6. RM: average number of rooms per dwelling
7. AGE: proportion of owner-occupied units built prior to 1940
8. DIS: weighted distances to five Boston employment centres
9. RAD: index of accessibility to radial highways
10. TAX: full-value property-tax rate per \$10,000
11. PTRATIO: pupil-teacher ratio by town
12. B: $1000(B_k - 0.63)^2$ where B_k is the proportion of blacks by town
13. LSTAT: % lower status of the population
14. MEDV: Median value of owner-occupied homes in \$1000's

- Your job is to estimate the Median Value (MEDV) for the newly redistricted neighborhoods.

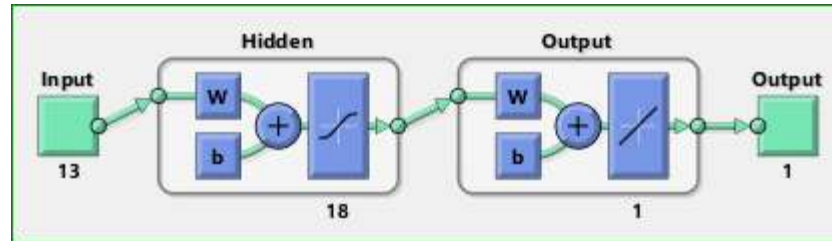
(*) Source: UCI Machine Learning Repository: <https://archive.ics.uci.edu/ml/datasets/Housing>

Case Study: House Values in Boston

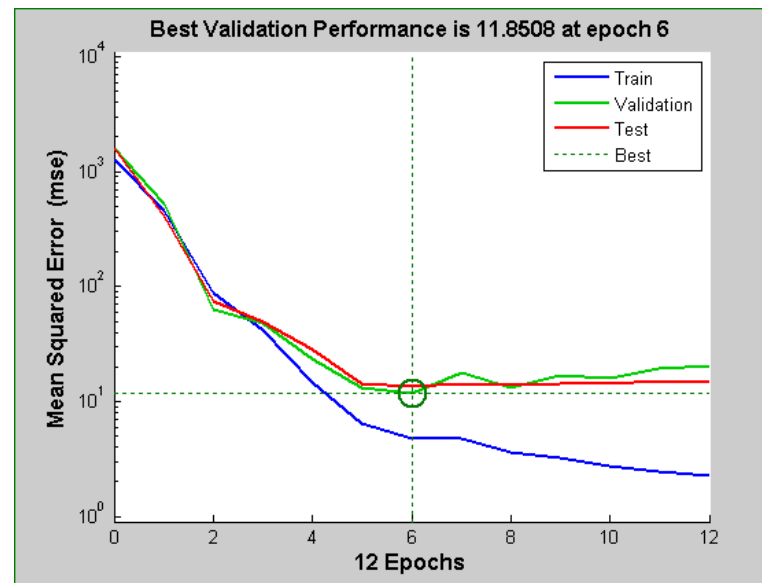
- You have 506 records with 14 fields, one of which is the desired target; *i.e.*, you have 13 attributes and the target, MEDV (Median House Values).
- You start out with a “large” feedforward network with 20 Hidden Sigmoidal Nodes and use Bayesian Regularization to train your network.
- After training several networks starting from different weight configurations, you note that, on average, the number of degrees of freedom decreases from 301 down to about 260.
- From this, you estimate that a network with 18 Hidden Nodes will suffice.
- You proceed to build a network with 18 Hidden Sigmoidal Nodes, and split your data into a Training set (80%), a Validation Set (10%) and a Test Set (10%).
- You train the network...

Case Study: House Values in Boston

- Network Architecture:

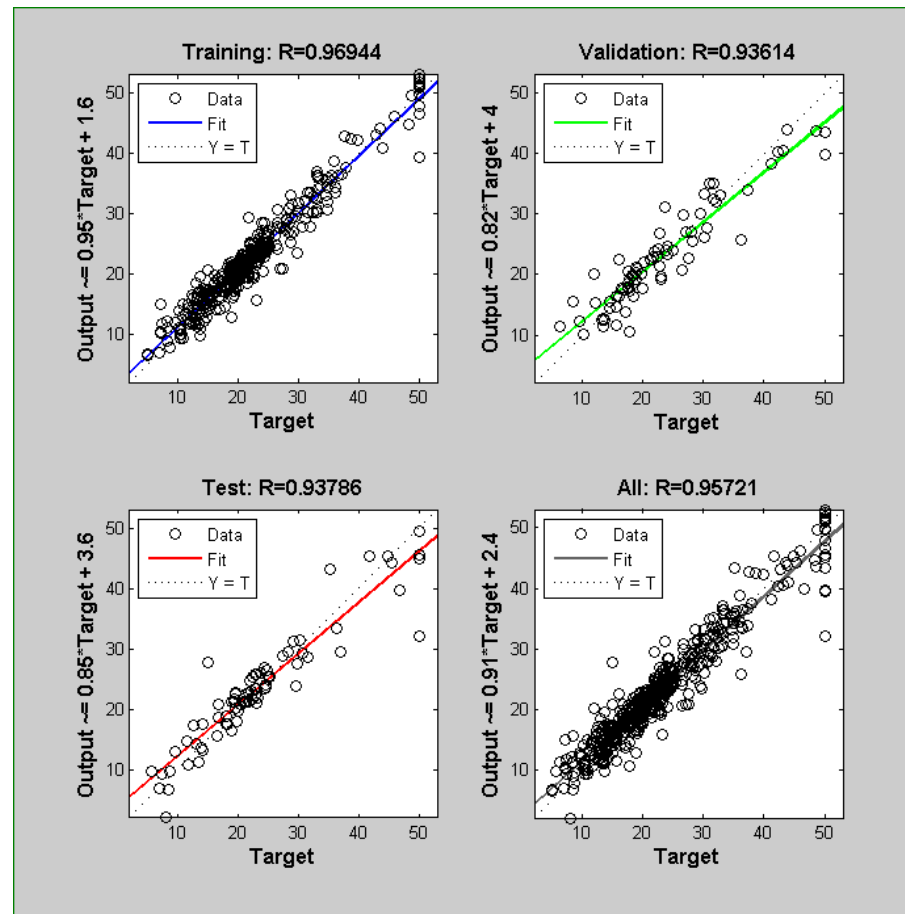


- Training profile as a function of the No. of Epochs:



Case Study: House Values in Boston

- Network Training, Validation, and Test Stats:



- You are now able to make predictions for individual neighborhoods...

Case Study: House Values in Boston

- Given any set of 13 attributes for any given redistricted neighborhood, you can use the above neural network to predict the Median House Value for that neighborhood.
- Furthermore, you assume that the R^2 obtained above for the Test Set, $R^2 \cong 0.88$, is close to the theoretically maximum R^2 , $\max R^2$, from which you deduce that the signal-to-noise ratio is around 2.7: $1/n \cong 2.7$.
- You also know the Standard Deviation of the Median House Values (the Target values, MEDV) is around \$9K, from which you deduce that your estimates' errors have a Standard Deviation of around \$3.1K, which is around 14% of the average house value (these data are from the '70s).
- You report your predictions for each neighborhood, along with your error estimates.
- You offer further services of building a bootstrap of several networks in order to generate more precise estimates and errors, a sensitivity analysis to find out the importance of each factor (for an added fee), and...
- You diplomatically remind the County Treasurer that Attribute 12 ("B") is very likely unlawful and unethical to use in any kind of county or governmental business. You offer to remove it and re-do the analysis (for an added fee).
- On to classifiers...

Neural Networks as Classifiers

- We already saw this in the case of AND, OR, XOR, etc.
- Feedforward NNs can be used as Classifiers.
 - Output = 0: “Company will go Bankrupt” (“Default on Bonds,” “Won’t Be Acquired,” “Trade is Unprofitable,” etc...)
 - Output = 1: “Company will be Solvent” (“Won’t Default,” “Will Be Acquired,” “Trade is Profitable,” etc...)
- Can be used for multiple classes; Output = 0,1,2...
- Ordinal Variables:
 - Output = -1 = “Out of the Money,” 0 = “At the Money,” 1 = “In the Money”

Variable Representation Tricks

- Binary Class Variables can be represented by 2 nodes:
 - Instead of representing “Class A” with 0 and “Class B” with 1, represent “Class A” with (0,1), and “Class B” with (1,0)
- Ordinal Variables can be represented with a “Thermometer Scale”
 - Instead of representing 1st, 2nd, 3rd, and 4th ordinals with 0, 1, 2, 3, we can use 3 nodes with (0,0,0), (0,0,1), (0,1,1), and (1,1,1).
- Continuous Variables can be represented through Interpolation across a several nodes:
 - *E.g.:* Use 4 nodes to interpolate a variable with range from 0-100 by interpolating 0-25 across nodes 1&2, 25-50 across nodes 2&3, etc.

Variable Representation Tricks

- Alternatively, Continuous Variables can be “Binarized”:
 - Turn on only Node 1 for values $[0,25)$, only Node 2 for $[25,50)$, etc.
- Periodic Variables can be Interpolated by keeping in mind that values in the extremes are similar
 - Could use 5 nodes to represent months. Dec-Mar would turn on nodes 1&2, Apr-Jun would turn on 2&3, Jul-Sep would turn on 3&4, Oct-Dec would turn on 4&5. Here nodes 1 & 5 overlap to represent continuity across Dec (and hence proximity across boundary).

Variable Representation Tricks

- These tricks are found to lead to more robust and faster convergence (Project topic?).
- Backprop has “less work to do” than if relied on a single weight to be modified every time a class output changes during the training instead of two or more.
- However, there is a tradeoff against model complexity (increased number of parameters) which could lead to overfitting.
- This should be used along with regularization, early stopping, etc. to avoid overfitting.
- Overall, these are good tricks because overfitting avoidance has to be used in any case.

Evaluating Classifiers

- Two-class classifiers (*e.g.* 0/1 or True/False) can make Two Types of Mistakes; In the language of Hypothesis Testing:
 - **Type I Error**: Assumes True when False (incorrectly rejects Null Hypothesis);
 - **Type II Error**: Assumes False when True (incorrectly fails to reject Null Hypothesis).
- Usually Type I Errors come at the expense of Type II Errors and vice-versa (there is a tradeoff).
- Tools for evaluating Classifier Performance:
 - Confusion Matrix
 - ROC Curve

(Aside: Type I vs. Type II Errors...)

- Biological Brains are far more biased towards Type I than Type II Error:
 - Far more likely to detect patterns out of randomness (Apophenia, Patternicity).
 - Far more likely to assume causal agency when there's none (Agenticity).
- Cost/benefit analysis shows this logical blind spot likely had adaptive advantage in biological evolutionary history; e.g.:
 - A noise in the bushes could be nothing, or a predator; Type I Error costs little, Type II can cost the animal's life.
 - Recognizing faces can be paramount to human survival, while seeing a face in the clouds costs little ...
- May explain superstitions, etc...

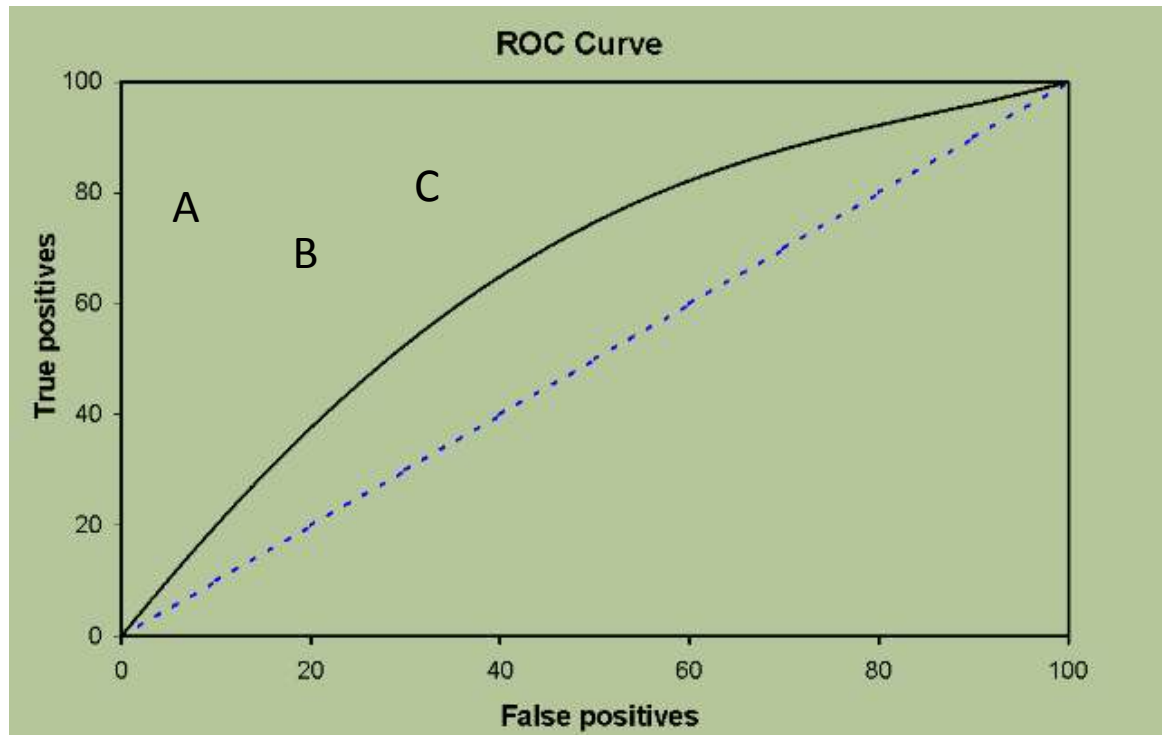
Confusion Matrix

- Quantifies predicted vs. actual classes.
- Quantifies error rates and correct classification rates.

		prediction outcome		
		p	n	total
actual value	p'	True Positive	False Negative	P'
	n'	False Positive	True Negative	N'
total		P	N	

ROC Curves

- “Receiver Operating Characteristics” (name came from use with radars, historically).



- Dashed line is “Random” classifier; Point (0,100) is the perfect classifier.
- The more “North-West” a classifier is, the better.
- Classifier A is objectively better than B or C, but it’s not clear that B is better than C.
- Can incorporate costs of errors into ROC curves to make them more relevant.

Questions, Comments...