REGRESSION

Caren Marzban
http://www.nhn.ou.edu/~marzban

Generalities

What you are reading is an expanded version of what is presented in my lecture at the short course. Expanded means that it has more words.

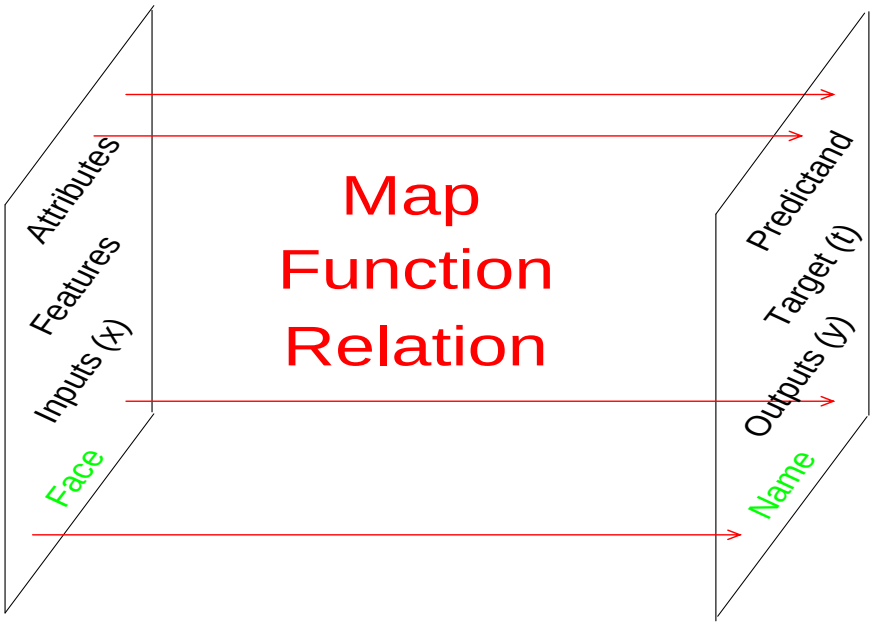
There may be statements in one talk that appear to contradict statements in some other lecture. If so, then it's probably either because the statements have been presented incompletely (without specifying all the contingencies and assumptions), for the sake of brevity, or because there is some controversy surrounding the issue. Either way, if you find some apparent contradiction, it probably means that you are grasping the material quite well.

You may come across some discussion over whether or not NNs are equivalent to traditional statistical methods. These discussions are mostly philosophical, though sometimes not. It's probably wiser to avoid the whole issue and use both NNs *and* more traditional methods all at the same time.

Regardless of whether NNs are equivalent to traditional statistics, NNs should definitely be considered a statistical tool. As such, all of the methods developed in traditional statistics have a place for application in NNs. There is no reason for NN researchers to reinvent the wheel. For this reason, my lectures are either based on or motivated by traditional statistics.

A Neural Network (NN) in both of my lectures refers to a feed-forward, multilayer perceptron. In particular, only two hidden layers (of weights) [1] are considered. Other types will be discussed in the other lectures. For example, The lectures by William Hsieh will discuss NNs with 3 hidden layer of nodes.

[1] Or equivalently, one hidden layer of hidden nodes.



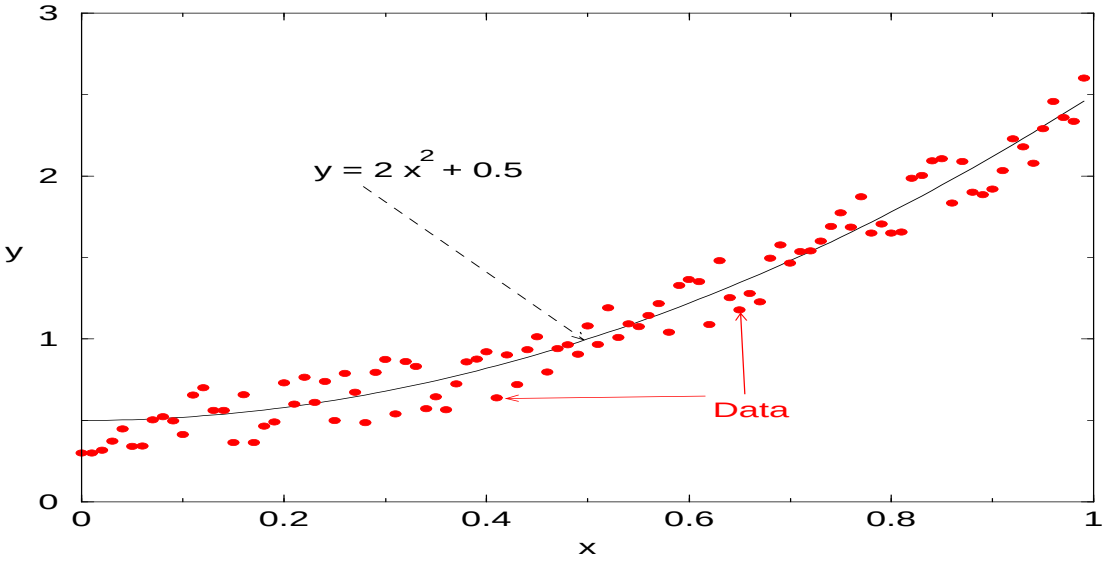
The whole business of NNs is about estimating the relationship between two sets of variables from data. Depending on the field, the two sets of variables are called by different names, and it's even common to confuse some of the names. I will talk only about the input variables (or predictors), and output **variables** (or targets). Output **nodes** are literally the outputs of the NN, intended to approximate the output variables.

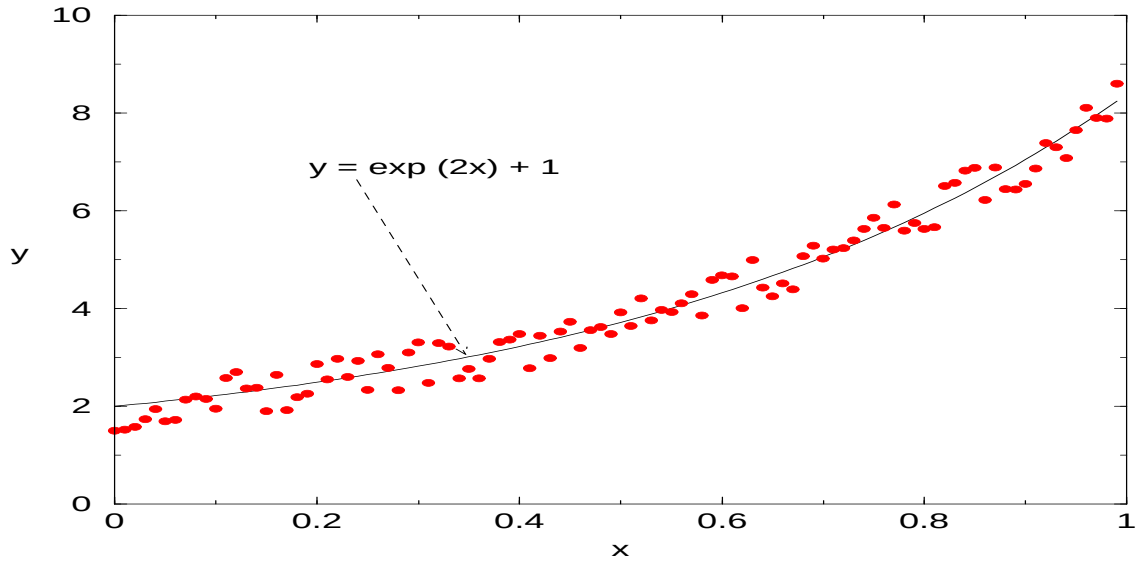
Talk 1: Regression.

By a "regression problem" we mean one where the output variables are continuous. For example, temperature, amount of rain, cloud height, hail size, etc.

Regression models come in two types: linear, and nonlinear. It's important to realize that those adjectives refer to the **parameters** of the model, and not the relationship between the input and output variables. For example, the top figure, below, is really a problem in linear regression, in spite of the nonlinearity of the underlying curve (solid). That's because one can simply square the values of x in the data, and then do linear regression between y and x^2 for estimating the parameters (i.e., 2 and 0.5). The bottom example, though, is a faithful nonlinear regression problem, because it cannot be linearized. (I'm waving my hands a little bit here, because I'm neglecting the way noise is handled in these models.)

Linear and nonlinear regression.

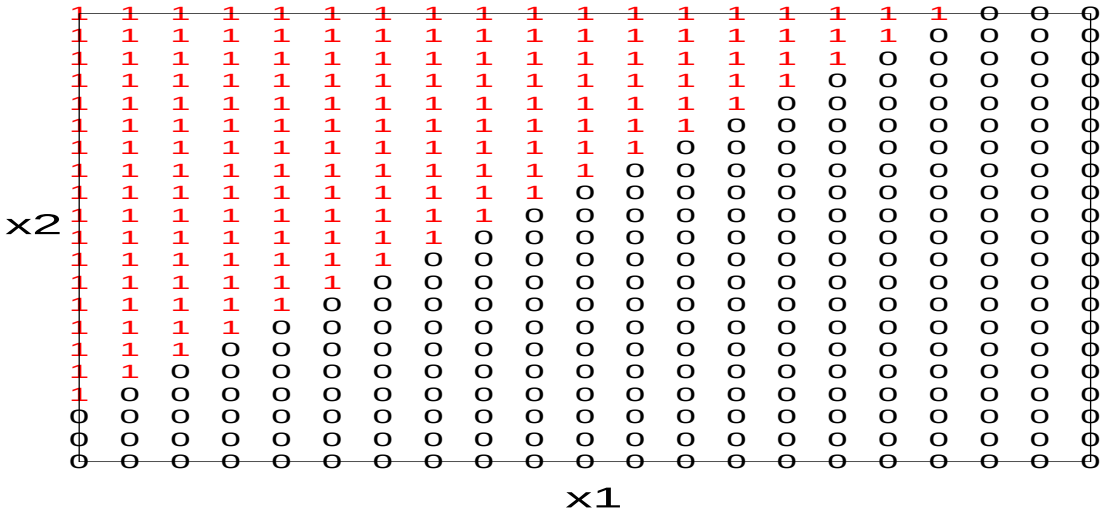


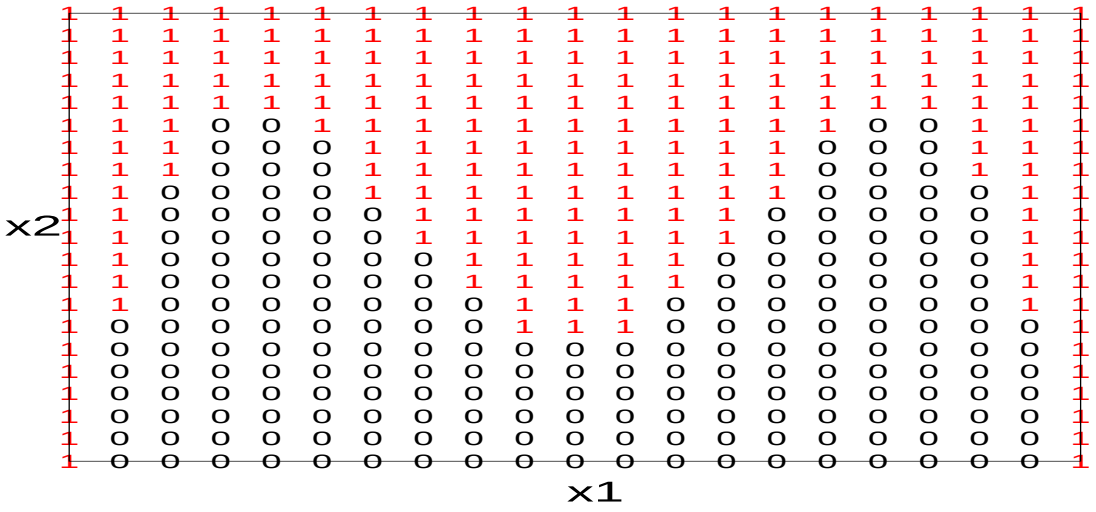


Talk 2: Classification (Discrimination).

As for a "classification problem," it's one where the output variables (targets) are discrete or categorical. An example of a categorical variable that has only two values (or classes) is tornado vs. non-tornado. A 3-class variable could take the values small, medium, or large. Of course, one has to "encode" these variables as outputs in an NN. For example, a 2-class variable can be encoded with a single output node taking values 0 and 1, respectively, for the two classes. Or it can be encoded with two 0/1 output nodes, with the node with larger value designating the class: (1,0)=class 0, (0,1)=class 1. Take a look at T. Masters (1993): *Practical Neural Network Recipes in C++*, Academic Press. If you have a problem with, say, 13 classes, or a variable that takes 13 values, is that a regression problem or a classification problem? Things can get ambiguous. And then you may come across something like logistic regression, which is a regression model for doing classification. Whereas in regression one talks about the underlying function, in classification the analog is the (decision) boundary. The two figures below, show examples of a linear and a nonlinear boundary. Note that x_1 and x_2 are both input variables, and the classes are labelled with 0 and 1. Anyway, classification is the topic of my second lecture.

Linear and Nonlinear decision boundaries.





Linear Regression

As I said, I will rely on traditional statistics. So, let's regress and consider simple linear regression.

The problem starts with data, and in case of simple linear regression they will be N pairs of numbers. One member of the pair is our input variable, x, and the other member is the output variable (target), t:

$$(x_i, t_i), \quad i = 1, 2, 3, ..., N$$

Then, we need a model, which in case of simple linear regression is

$$y(x, \omega) = \omega x + \theta \quad ,$$

and we are supposed to estimate the parameters ω and θ from the data. This model assumes that our data satisfies

$$t_i = \omega x_i + \theta + \epsilon_i \quad ,$$

where ϵ_i is some amount of error (or disagreement) that we can tolerate between the target t_i and the output $y(x_i)$.

We need one more thing. Something that quantifies what we mean by "tolerate". A common example is the mean square error (MSE),

$$E(\omega) = \frac{1}{N} \sum_i^N \epsilon_i^2 = \frac{1}{N} \sum_i^N [y(x_i, \omega) - t_i]^2.$$

And now we want to find the values of the parameters that minimize this error. So, we look for solutions to the following pair of equations:

$$\frac{\partial E}{\partial \theta} = 0 = \frac{\partial E}{\partial \omega}.$$

It's straightforward to show that the minimum occurs at (homework)

$$\omega = \frac{\overline{xt} - \overline{x}\,\overline{t}}{\overline{x^2} - (\overline{x})^2},$$

where $\overline{\ }$ represents ordinary average of the what's inside. E.g.,

$$\overline{xt} = \frac{1}{N} \sum_i^N x_i t_i \qquad \text{etc.}$$

So, this ω - The so-called Ordinary Least Squares (OLS) estimate - can be computed (estimated) directly from the data. There is a similar expression for the OLS estimate of θ. (homework)

By the way, the linear correlation coefficient between two variables x and t is (also see Appendix B)

$$r = \frac{\overline{xt} - \overline{x}\,\overline{t}}{\sqrt{(\overline{x^2} - (\overline{x})^2)(\overline{t^2} - (\overline{t})^2)}},$$

and is not equal to the regression coefficient ω.

NN

Now, let's talk about NNs. In terms of an equation an NN is simply a generalization of the equation $y = \omega x + \theta$:

$$y(x, \omega, H) = g\left(\sum_{i=1}^{H} \omega_i \; f(\sum_{j=1}^{N_{in}} \omega_{ij} x_j - \theta_j) - \omega\right)$$

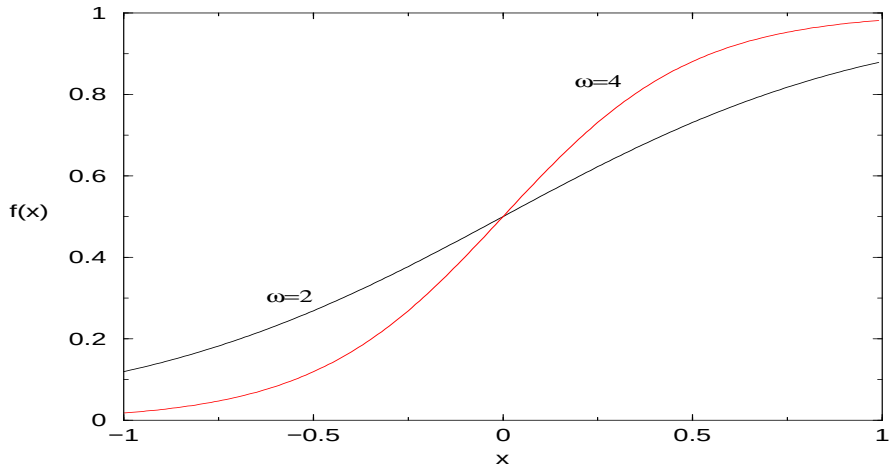
where H is the number of hidden nodes, $f(x)$ is some sigmoidal function, and $g(x)$ is either a sigmoid or a linear function. (See the previous lecture.) So, in addition to estimating all the parameters in this equation, we also have to come up with a good value for H.

It is often said that H is a measure of the nonlinearity of the relationship between the input and output variables. The fact is that both H **and** $|\omega|$ carry that burden. This can be seen by examining the figure below. It plots the logistic function

$$f(x) = \frac{1}{1 + e^{-\omega x}}$$

for $\omega = 2$, and 4.

It's evident that if the magnitude of ω is "small" (like 2), then $f(x)$ is linear in x, and the whole NN equation becomes linear in x. In other words, if the weights of the NN are small, then the NN is a linear function, regardless of how many hidden nodes it has (i.e., it's linear regression). At the same time, for large $|\omega|$ (like 4), $f(x)$ is nonlinear in x. Then, even for a moderately small number of hidden nodes, the NN can end up being quite nonlinear. Of course, "small" and "large" are relative to the size of x.



Why all the concern over nonlinearity? Because too much nonlinearity can lead to overfitting, which in turn leads to poor predictions and poor performance. Overfitting refers to the situation where the fit to the data is more wiggly than it should be. Of course, we don't know what the right fit is, but there are different techniques for trying to estimate it. One sure way is to check the performance of the model on completely new (independent) data. Overfitting will almost certainly lead to worse performance on the new data. The next few pages will tell you how to go about training an NN that does not overfit. Actually, there is always a chance that an NN under- or overfits. We can only try to minimize that chance.

$\omega =?$

Estimating the weights is probably the best-studied aspect of NN development. Many methods though appear ad hoc. Many start from some almost arbitrary expression for an error (or objective) function (like MSE) and proceed to minimize it. Although the minimization idea is quite intuitive, it's useful to keep in mind that a minimization of an error function typically corresponds to the maximization of a probability function. The argument goes something like this.

If we want to estimate ω, it makes sense to take it to be the most probable value, given the data D. Writing the probability of any ω, given data, as $P(\omega|D)$, the basic laws of probability imply that it can be written as

$$P(\omega|D) \sim P(D|\omega) \times P(\omega),$$

where $P(D|\omega)$, called the likelihood, is the probability of getting the data, given the weight. $P(\omega)$ is the probability of having a weight ω before we have even seen the data. The likelihood can be computed from the data, while the latter expresses any belief or constrain we may have regarding the weights (independent of the data at hand).

Given that they are both probabilities, they can be written as exponentials:

$$P(D|\omega) \sim e^{-E_D(\omega)} \quad p(\omega) \sim e^{-E_W(\omega)},$$

which means

$$p(\omega|D) \sim e^{-[E_D(\omega)+E_W(\omega)]} \equiv e^{-E(\omega)}.$$

So, the maximum of $P(\omega|D)$ occurs at the minimum of $E(\omega)$. In other words, the most probable ω, given data, minimizes $E(\omega)$.

For example, if data are so-called gaussian or normal, i.e.,

$$P(D|\omega) \sim e^{-[y(\omega)-t]^2} ,$$

and our uncertainty regarding the weights is also gaussian, i.e.,

$$P(\omega) \sim \exp^{-\omega^2} ,$$

then

$$E(\omega) \sim \Sigma[y(\omega) - t]^2 + \Sigma\omega^2 .$$

The first term is nothing but MSE, and now we have a new term which is called the weight-decay term. The role of the weight-decay term is to prevent the weights from getting too large. As such, it is highly recommended to instruct your favorite NN simulation package to include such a term, because as we said before, large weights can lead to overfitting. Again, I'm waving my hands a little bit, because I've neglected the coefficients of the two terms. The relative size of the two terms is important because it quantifies again what we mean by weights that are "too large". What is gained, though, is that we now have only one number to decide on (the ratio of the two coefficients) instead of the size of every single weight in the NN. And chances are that your NN simulation package will offer you some wise choices for the coefficient of the weight-decay term based on some theoretical or empirical criteria. Just go ahead and include the weight-decay term in the error function, and if you really want to know what exactly the criteria are for selecting its coefficient, get on the web and look for "weight decay". Needless to say, this coefficient determines the *overall* size of all the weights. It doesn't say anything about the relative value of each weight.

Finally, note that the above exposition exposes a crucial assumption often unmentioned in NN circles. Specifically, note that even in NNs the very choice of the error function to minimize implies

an assumption regarding the underlying distributions of the data. For example, the choice $E =$ MSE assumes normality. Be skeptical of claims of NNs as assumption-free statistical tools.

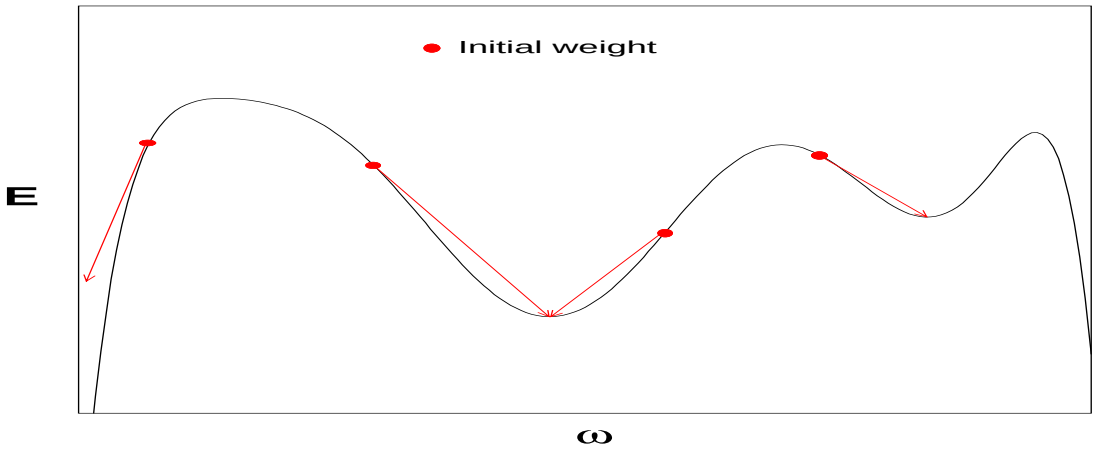
Appendix A will also show that the choice of the error function E imposes an interpretation on the output nodes of an NN. Another instance of this type of output interpretation will be given in the next lecture.

Now, let's return to the minimization task. Given the nonlinearity of $E(\omega)$ on ω, it is impossible to solve the $\partial E/\partial\omega = 0$ equations exactly. The usual approach is to adopt some iterative approach that converges on the minimum of $E(\omega)$. In NN circles these are called training or learning algorithms. The brute force approach could be something like this: Vary all the weights over their full range in some small increments. Calculate the E at each increment, and then select the weight values that yield the lowest E.

The good news is that there exist much smarter ways of finding the minimum of E. Let me just mention some of their names: gradient decent, backpropagation, (scaled) conjugate gradient, simulated annealing, genetic algorithms, etc. There is a large body of literature on these optimization techniques. Sue Ellen Haupt will speak about the latter. If you i like programming, then I invite you to write your own code. But it's probably more practical to just download the code from somewhere. A very useful site is ftp://ftp.sas.com/pub/neural/FAQ.html .

The fly in the ointment is that there almost always exist (too) many local minima. In other words, if you pick some random initial weights and start training (i.e., minimizing E), you'll find that after some time E doesn't seem to decrease any further. You might think that you have found a minimum. However, if you start training from different random initial weights, you'll probably find a different minimum - different both in the value of E and the corresponding final weights. These are all local minima. See the figure below.

There are many methods for both avoiding local minima and escaping them, but none of them guarantee that a global minimum has been found. Actually, some of them guarantee a global minimum, but only in theory. One of my favorites that works well in practice is simulated annealing (see Masters, referenced above). In practice, one doesn't even need to find the global minimum anyway; a sufficiently deep local minimum usually suffices. However, to assess what "sufficiently deep" means, it's better to decide for yourself based on your own data. Just train from many (say, 5, 10, or 100) different initial weights, and plot the distribution of the final E's. This will show you not only the most likely local minima but also the likelihood of landing in a global minimum.



H =?

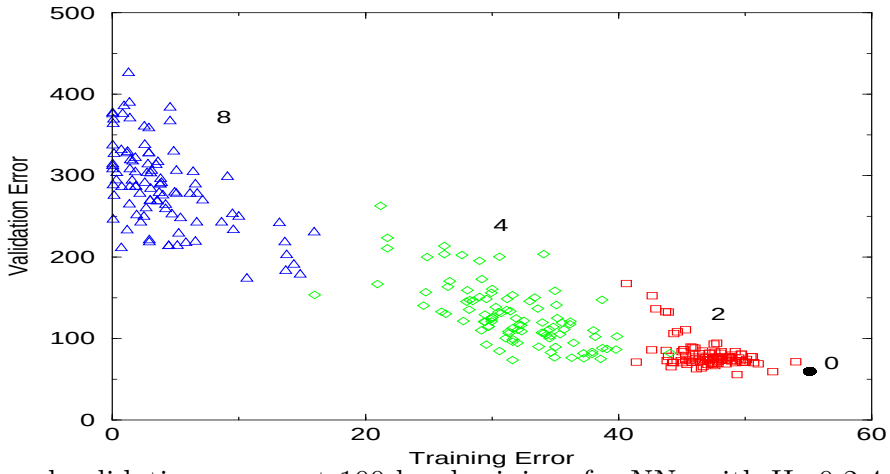
Now, how do we decide on the number of hidden nodes H?

Again, there are many statistical methods that can be adapted to answering that question. Two popular ones are cross-validation and bootstrapping. They are both examples of resampling methods. In their simplest form, one repeatedly trains with subsamples of the data (i.e., training sets), and the optimal NN is selected to be the one with the lowest *average* error over the unused subsamples (the validation sets). From the variance (over the subsamples) of the validation errors one can construct a confidence interval for the performance of the NN.

In what proportion should the data be partitioned into training and validation? There are some theoretical results that suggest the 2/3 rule, namely that 2/3 of the data should go in the training set, and the remaining 1/3 in the validation set. (See B. Efron, and R. J. Tibshirani 1993: *An Introduction to the Bootstrap.* Chapman & Hall.)

Technically, one needs a third set, which is sometimes called the test set. The reason is that the performance of an NN on either the training or the validation set is an optimistically biased estimate of performance. It is the performance of an NN on the test set which is unbiased.

To find the optimal number of hidden nodes, one cannot forget the local minima. The following figure shows what I call a tv-diagram. The x-axis displays the training errors obtained at 100 local minima, and the y-axis plots the corresponding validation errors. Both are calculated for $H = 0$, 2, 4, and 8.



Training and validation errors at 100 local minima for NNs with H=0,2,4,8. (tv-diagram)

There is a wealth of information in tv-diagrams. Let's just talk about a few useful features. The first thing one notes is the increasing spread (in both x and y directions) of the points as one increases the number of hidden nodes. You can imagine the devastation that local minima can bring about if H is even larger. The next thing a tv-diagram does is to expose the fallacious practice of choosing H to be the one that yields the lowest validation error. In the figure below, for example, there is one point in the $H = 2$ cluster of points that has lower training and validation errors than the $H = 0$ NN, suggesting that the optimal H should be 2. However, that point does not have the lowest training error in the $H = 2$ cluster, and so is only a local minimum of the $H = 2$ NN. In fact, the point with the lowest training error in the $H = 2$ set (i.e. the global minimum of that NN) has a higher validation error than the $H = 0$ set. As such, the global minimum of the $H = 2$ NN overfits the data, implying $H_{optimal} = 0$.

Below, I will show one more thing that is wrong with picking H according to the lowest validation

error, namely that an NN with the lowest validation error actually overfits the *validation* set itself.

Let me mention in passing that the above tv-diagram is based on actual data and it's not something I concocted to make a point. In this case, the best NN was one with $H = 0$. And when I examined the magnitude of the weights, they were all in a range where the logistic function is linear (look at the plot of the logistic function a few pages ago). As such, the best NN was nothing more than linear regression! The point is that its wise to not neglect the possibility that things may be linear anyway, in which case not only linear regression will do just as well, but given the linearity of linear regression there is a number of extras that you can get from it (like standard errors of the weights) that are harder to get for NNs.

There is a lot more information in tv-diagrams. Let me just suggest that you look at the distribution/histogram of the training and validation errors for the different H clusters in your own data.

Everything I said above amounts to filling in the elements in the following table. "seed" refers to different initial ω's, and (t,v) refers to (training error, validation error).

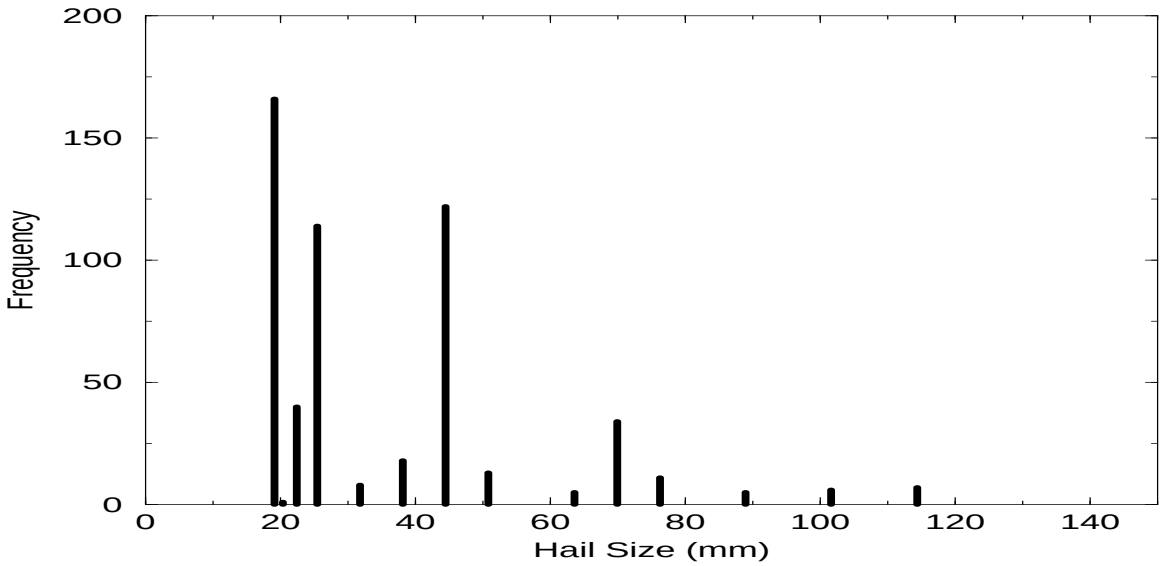
H	Seed	Bootstrap Trial		
		1	2	...
2	1	(t,v)	(t,v)	...
	2	(t,v)	(t,v)	...
	...	(t,v)	(t,v)	...
4	1	(t,v)	(t,v)	...
	2	(t,v)	(t,v)	...
	...	(t,v)	(t,v)	...
8	1	(t,v)	(t,v)	...
	2	(t,v)	(t,v)	...
	...	(t,v)	(t,v)	...
	...	...	...	...
	...	tv-diagram	tv-diagram	...
		↓	↓	
		H_1	H_2	$\rightarrow H_{\text{optimal}}$

Each column yields a tv-diagram. Each tv-diagram suggests an optimal value for H. And given, say 17, bootstrap trials, we'll have 17 suggested optimal values for H (the H_i in the table). If you are lucky, they'll all be the same. Often, though, they are not, and so you have one more decision to make regarding which of the H_i to pick as the final optimal one in the NN that you will be fielding, turning in to your boss, or selling. If you have performed enough bootstrap trials to have enough H_i, plot a histogram for them, then go for the mode, i.e., the most frequent H_i. Otherwise, go for the median. The final H should be the one that has the lowest *average* (over the bootstrap trials) validation error. If it's not, suspect local minima doing their bad thing.

Usually, when you hear about overfitting data, it's the training data that is being talked about. As I alluded to above, there is one common mistake that people often make that leads to overfitting the validation data. Note that each column of the table relies on only one partitioning of the data into training and validation. So, in the first column (trial = 1), there is only one training and one validation set. H_1 is the number of hidden nodes that yields the lowest error on that specific validation set. If we select H_1 as our optimal value of H, we have just guaranteed that our NN performs bet on that validation data. But that validation data is only one specific manifestation of all the data, and so, chances are that the resulting NN will not perform optimally in the real

world. In this sense, our model overfits that particular validation set. Similarly, $H = H_i$ overfits the validation set in the i^{th} bootstrap trial. It does not matter whether we are overfitting the training set or the validation set - either way, our generalization performance will be poor. So, make sure you go through the whole table and select $H_{optimal}$ as the final one.

Practical Application - Hail Size



The distribution of hail size.

We are given 15 variables that are thought to be related to hail size, and you have data (550) on all 16 (15+1) variables. Our output node is then supposed to represent hail size. Almost always some amount of preprocessing is called for. If you have infinite data, you probably don't need to worry about any of these. Otherwise, a few good things to do are as follows:

Examine the distribution/histogram of all the variables (input and output). Chances are you'll learn things about the data that will help in better designing an NN. The least you should do is to exclude the outliers from the training set. Outliers are often pathological cases, and including them in the training set can cause the NN to learn the wrong lesson. For example, it is well-known in linear regression that OLS estimates (i.e., parameters that minimize mean square error) are very sensitive to the presence of outliers. Of course, if you have established that your choice of error function is not sensitive to outliers, then never mind. Finally, there are times when outliers are exactly the cases you are interested in; for example, that once-in-a-century earthquake. If so, then never mind what I said about excluding outliers.

Transform all the input and output variables to z-scores *a la*

$$z = (x - \mu)/\sigma,$$

where μ and σ are the average and standard deviation of the variable. This will assure that all of the variables have a mean of zero and a standard deviation of 1. There are many reasons for why this is a good thing to do. Suffice it to say that it will make sure that all the variables vary over the same range of values.

Exclude collinear inputs. Generally speaking, including superfluous inputs in an NN does not adversely affect performance. However, any additional input node introduces a whole new set of weights that must be estimated. This will just increase the chances of overfitting. So, if you have two inputs that are highly correlated (i.e., the r between them is "high"), then input just one of

them. How large is large, will again depend on your data. Certainly, if you have two variables that have a correlation of, say, $r = 0.98$, chances are they are measuring the same thing (modulo noise). Exclude one of them.

Do a few scatterplots of the various input variables. Again, you'll probably learn some useful things. For example, you may find that there is a clear nonlinear relationship between two variables. And when you look at exactly what those variables are, you'll see that one is measured on a log-scale. Then, when you look at the scatterplot again, with one of the axes on a log scale, you get a perfectly linear relationship with say, $r = 0.97$. So, you have just identified another superfluous variable that is better to exclude from the NN. In the hail example, this reduces the number of inputs from 15 to 9.

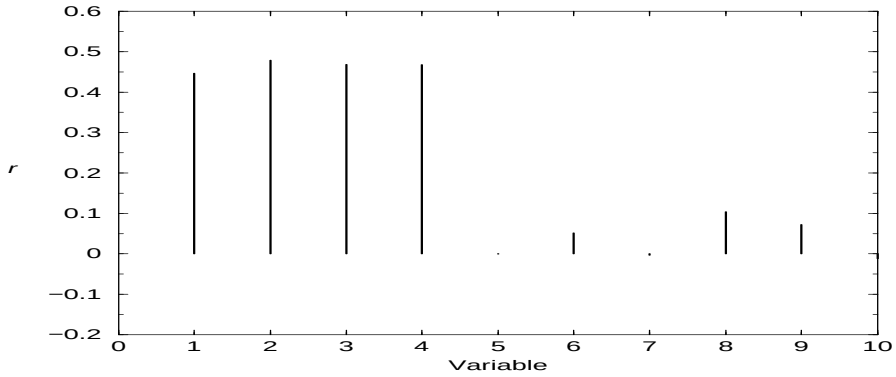
Choose the error function wisely. For example, if you suspect the errors are gaussian, then MSE is a good choice for a regression problem, especially since it will assure that the output of the NN has a nice interpretation, like the conditional average of the output variable, given the inputs (Appendix A). If you do assume normality, make sure that after you've trained your NN and all is done, that your assumption was not violated. The way to do that is to check the normality of the residuals $(t_i - y(x_i))$. Just plot the distribution/histogram of the residuals and at least visually check that it is bell-shaped.

There are more things that you can do in helping out the NN in learning the right function, but like many other things, it all depends on the data. So keep an open eye for things that can simplify the learning task. At the same time, don't over simplify!

By the way, the optimal number of hidden nodes for the hail NN turns out to be 4. And for those who care, the 9 inputs into the NN are:

1	Cell-based vertically integrated liquid
2	Severe hail index
3	Storm-top divergence
4	Midaltitude rotational velocity)
5	Height of the wet-bulb zero
6	Height of the melting level
7	Vertically-integrated wet-bulb temp
8	Wind speed at the equilibrium level
9	Storm-relative flow at $-20°C$ level

One thing you don't want to do (and I mention it because some do it) is to use the linear correlation between the output variable and each of the input variables to select the input variables that should be used as inputs to the NN. Actually, it's good to calculate this r because it is probably the only unambiguous measure of predictive strength you can assign to the different input variables. For example, the following figure suggests that variables 1 through 4 are "stronger" predictors than the other variables. But "stronger" refers to *linear* strength, because r is a measure of linear correlation. Selecting only the variables 1 - 4 will implicitly linearize the data. The NN has thus been rendered moot; you might as well have just used linear regression. In fact, in the hail size case, the inclusion of variables 5 - 9 as inputs to the NN improves performance.

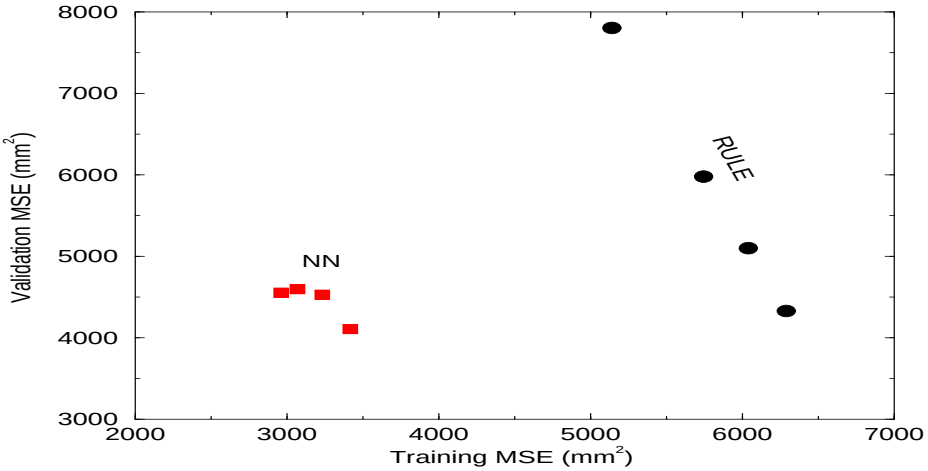


Look at r, but do not select inputs based on r.

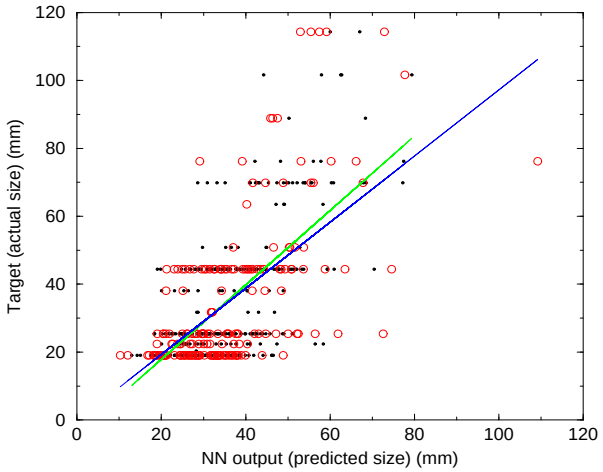
Performance

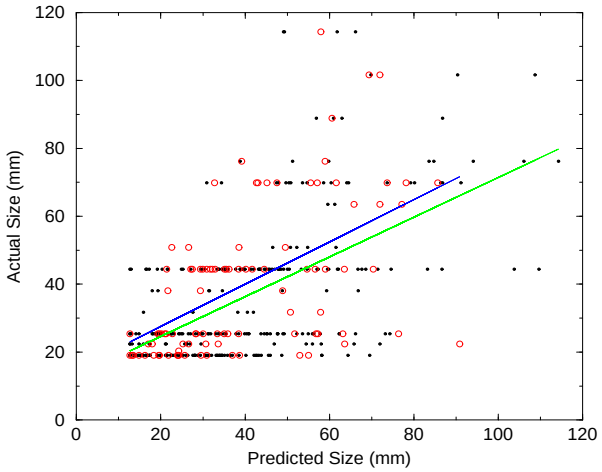
The issue of performance is a very thorny one, especially when you have to compare the performance of two different models to decide which one is better. I will talk a bit more about this matter in my lecture on performance assessment. The most important lesson, in the words of the late Allan Murphy, is that performance is a *multifaceted* concept. It is entirely possible that one model outperforms another model in terms of one measure of performance but not in terms of another. That's why it's important to choose the measure(s) of performance thoughtfully. Unfortunately, that's easier said than done. In many situations/problems it's difficult to decide what the "right" measure(s) should be. And that's why most people just pick measures with some nice and simple properties and go from there. Given that approach, the least we can do is to make sure that the choice of the measure does not contradict the assumptions of the model or properties of the data. (By the way, an error function, like MSE, is also a performance measure.) The situation is even worse, because even though performance is a multifaceted thing, we frequently end-up minimizing (or maximizing) a single scalar quantity (like MSE). To partially compensate for this last shortcoming, I will try to express performance in terms of figures and diagrams, instead of single numbers. And I recommend that you do the same.

Then again, there are times when you are almost forced to produce results in terms of single, scalar measures. This is especially true when you have to present your results to people who don't acknowledge the multidimensionality of the problem. Or when you have to produce some dramatic means of showing that one method is better than another. For example, I want to show that the hail size NN outperforms the older algorithm (which is a heuristic rule). So, I'll show the following figure, showing the training and validation MSE values of the two models for four different Bootstrap trials. It's not too bad. It shows that not only the NN is better in both the training and validation errors, it is also more "consistent" in treating the four bootstrap partitions of the data. The latter is evident from the relative scatter of the points.

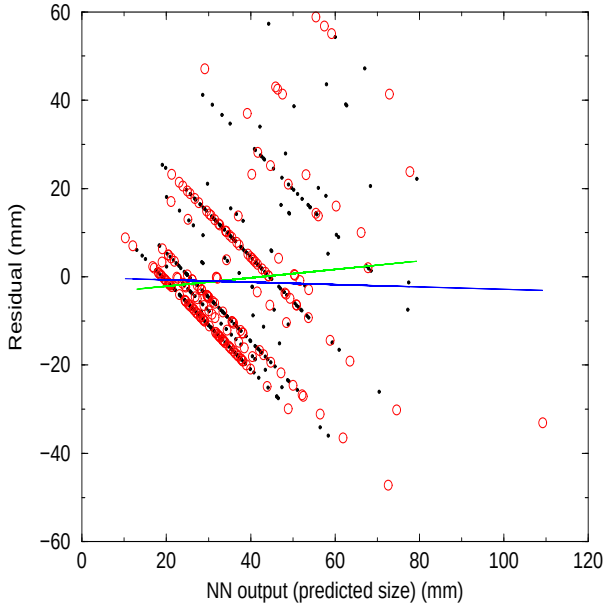


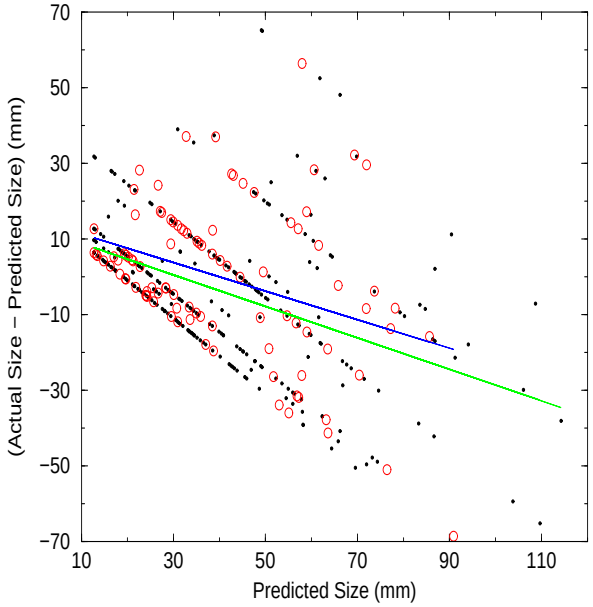
Having gotten that out of the way, then you can concentrate on more multidimensional measures, like figures and diagrams. The following figures show the scatterplots for the NN (right) and the older rule (left). The black dots are the training data, and the red circles are the validation data. What we would like to see is a small scatter of points, close to the diagonal line. We can see that not only the NN has less scatter than the rule, the points are also more along the diagonal. This becomes evident if we plot the regression lines through the training and validation points. Keep in mind, though, that these lines are only for providing a visual means of assessing performance; as regression models of actual data, they violate the assumptions of regression.



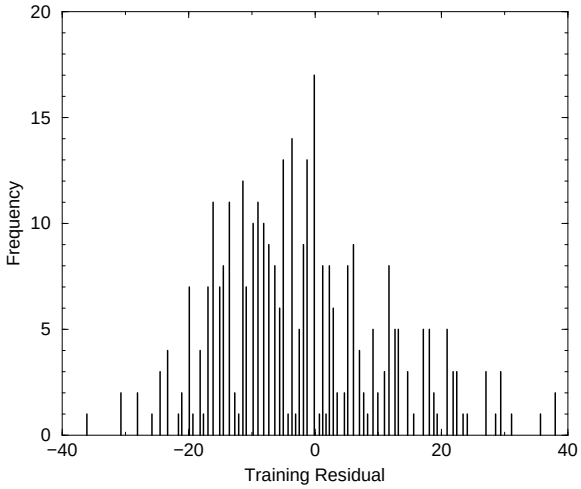


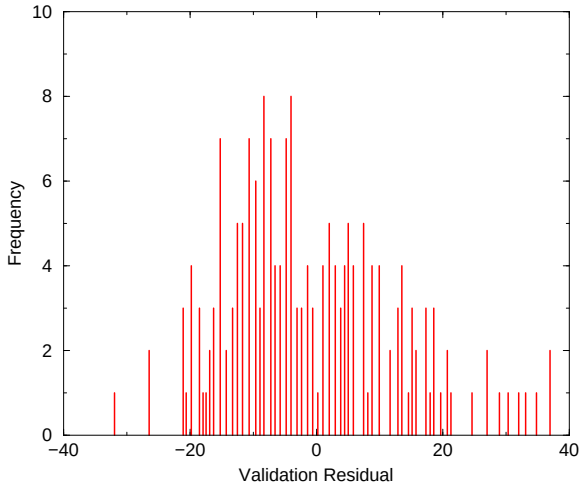
Another facet of performance is captured by residual plots. These are shown below. This time, what we would like to see is a scatter of points tightly arranged around the horizontal line where the y-axis is zero. Regression fits through these points again provide a visual illustration; the NN fits for both the training and validation data are right along the x-axis at around y=0. By contrast the Rule fits are clearly unsatisfactory.





One last thing we want to do is to make sure that the errors are normally distributed, because we did end-up training the NN by minimizing MSE. Of course, we don't know the actual errors, but we do have the residuals. So, just check that the residuals are normally distributed, or at least bell-shaped. The following figures show that both the training and the validation residuals have bell-shaped distributions. It's not really necessary to check normality exactly, unless you are planning to do some serious hypothesis testing.





Finally, while we are on the topic of performance (of a regression model, NN or otherwise), search for "Bias and Variance" on the web. You'll see that in many problems the task of coming up with the "best" model is a constant struggle between two opposing forces - bias and variance; different "optimum" models amount to different balances between the two. I will briefly mention this in my lecture on performance assessment.

Discussion

We have learned how to develop an NN properly. We have also developed an NN for predicting hail size and shown that it outperforms the older model (referred to as "rule"), at least in terms of MSE, and a few other multidimensional measures.

However, try to avoid model comparison altogether. There are just too many loop holes. (Just for fun, check out "No free lunch theorems" on the web.) For instance, suppose I come to you and submit that my model (A) is better than yours (B). Well, the first thing you should ask is whether A is better than B in terms of *all* the relevant measures of performance. The answer is probably in the negative, which means that your model is still better in some facet of performance. Then, on what data were the two models tested? Perhaps, my model outperforms yours (in some sense) on one data set but not on another. Are the performance numbers for the two models given with error bars? In other words, it could be that my model only looks like it's better, but not in any statistically significant sense. The list of questions is too long to get into here, but you get the picture. Just avoid that can of worms, if you have the luxury to do so.

Try to use graphical (or multidimensional), rather than scalar, means of assessing performance. Scatterplots and residual plots are just two examples. If you have to distill performance into one or two numbers, make sure you make a clear declaration that you have done so and that some information is apt to have been lost.

The linear correlation coefficient of the points on a scatterplot, r, is one scalar measure of performance. Any regression book will prove that r^2 is the percentage of total variance explained by the model (NN or otherwise), and so $r \sim r^2 \sim 1$ is good. The r^2 of the training and validation sets for the NN are 0.51 and 0.40, respectively. By comparison, the same quantities for the rule are 0.34 and 0.29. (These are just for one bootstrap trial.)

Any nonlinear pattern in residual plots suggests that the NN has learned the wrong function. What you want to see is a random pattern without a clear linear or nonlinear pattern. You may notice a "fanning" pattern with the points residing on a series of parallel lines with slope=-1. The fanning pattern is not too good because it implies that larger hail sizes are less accurate. But that's just life. The parallel lines, however, are just a consequence of the fact that one is plotting (target - output) vs. output, and that "output" appears on both sides. That's nothing to worry about.

There exist many corrections to these r's accounting for nonlinearity, no. of weights, etc. (see N. R. Draper and H. Smith, Applied Regression Analysis, John Wiley & Sons, 1981). But if you go as far as I have suggested in expressing performance carefully, you should be in the safe zone.

Finally, it is very tempting to try to see what the NN is doing. At the simplest level, you may be tempted to find out which of the inputs the NN is concentrating on for making predictions. All I can say is don't! It is almost impossible to diagnose the weights of an NN. Appendix B will show some of the reasons for this warning.

I cannot end this lecture without at least mentioning the "curse of dimensionality." One may ask "Why NN, and not some other nonlinear model, like polynomial regression?" After all the space

of polynomials is huge and probably includes the function underlying the data at hand. The answer is, again, "Overfitting." The number of parameters (weights) in polynomial regression grows exponentially with the number of input variables (homework). With too many parameters there is a higher chance of overfitting the data. By contrast, the number of weights in an NN grows only linearly with the number of inputs. What's interesting is that in spite of this timid growth in the number of weights, an NN is quite capable when it comes to fitting (almost) any function to any desirable accuracy. In short, NNs are small enough to not overfit as badly as some other models, but big enough to be able to learn (almost) any function. (See Bishop, referenced below).

Appendix A

This appendix is based on C. M. Bishop, "Neural Networks for Pattern Recognition", Clarendon Press, Oxford (1996), Chapter 6. Here we show that if E is chosen properly, then the output of a regression NN can have a desirable interpretation.

As shown above, for gaussian noise, the right error function to minimize is MSE:

$$E = \frac{1}{N}\sum_{i}^{N}[y(x_i,\omega) - t_i]^2$$

As $N \to \infty$, we can replace the sum with integral:

$$\begin{aligned} E &= \int [y(x,\omega) - t]^2\, p(t,x)\, dt\, dx \\ &= \int [y(x,\omega) - t]^2\, p(t|x)\, p(x)\, dt\, dx, \end{aligned}$$

where $p(t,x)$ is the joint probability of the input and the target, and by definition $p(t,x) = p(t|x)\, p(x)$. Using the following equations

$$< t|x >= \int t\, p(t|x)\, dt$$

$$< t^2|x >= \int t^2 p(t|x)\, dt$$

$$\begin{aligned} (y-t)^2 &= (y- < t|x > + < t|x > -t)^2 \\ &= (y- < t|x >)^2 \\ &+ 2(y- < t|x >)\,(< t|x > -t) \\ &+ (< t|x > -t)^2 \end{aligned}$$

one can write

$$\begin{aligned} E(\omega) &= \int (y(x,\omega)- < t|x >)^2\, p(x)\, dx \\ &+ \int (< t^2|x > - < t|x >^2)\, p(x)\, dx. \end{aligned}$$

The second term is independent of ω. Therefore, the global minimum of $E(\omega)$ occurs at $\omega*$ when the first term vanishes, i.e.

$$y(x,\omega*) =< t|x > \quad .$$

In words, the output node of a (well-) trained NN is simply the conditional average of the target, given the input. This is very nice and just what we would want from a regression model. Think about it!

Appendix B

In this appendix I will illustrate some instances in which the weights of a model have no interpretation. One situation is where two (or more) of the input variables are correlated. This is referred to as collinearity.

Consider linear regression with two input variables, x_1, and x_2. Suppose we employ only x_1 to model the dependent variable,

$$y = a_1 x_1 + b_1,$$

and suppose we find $a_1 = 0.7$ as our OLS estimate of a_1 (see text, above). Now, do the same with x_2:

$$y = a_2 x_2 + b_2,$$

and suppose we find $a_2 = 0.3$ as the OLS estimate of a_2. Finally, consider both variables at once;

$$y = \alpha_1 x_1 + \alpha_2 x_2 + \beta.$$

It is simple to show that the OLS estimates for α_1, and α_2 satisfy

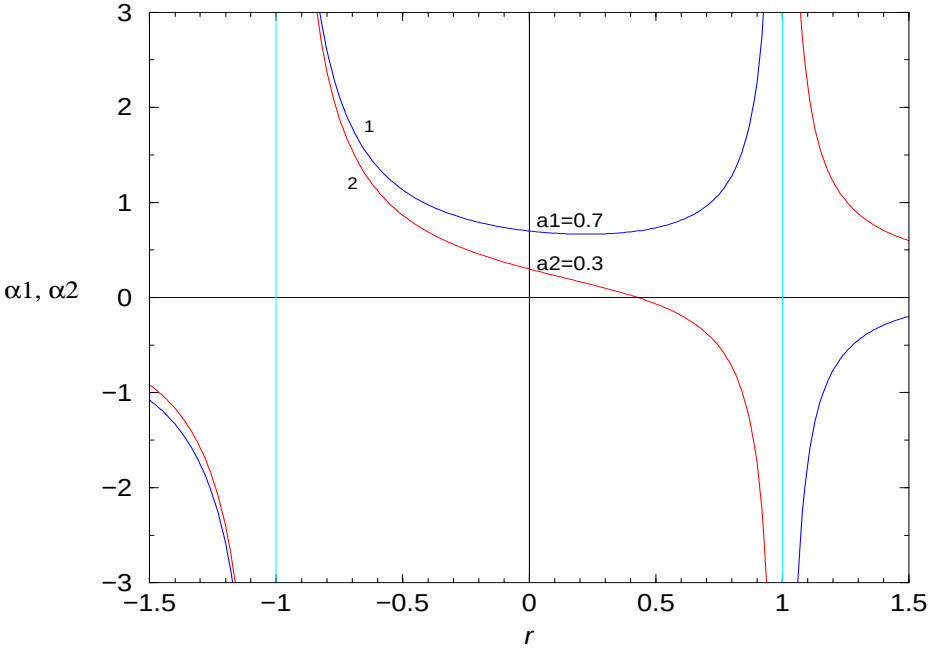
$$\alpha_1 = \frac{a_1 - r\, a_2}{1 - r^2} \qquad \alpha_2 = \frac{a_2 - r\, a_1}{1 - r^2},$$

where r is the correlation coefficient between x_1 and x_2. It is defined as

$$r = \frac{\overline{x_1 x_2} - \overline{x_1}\ \overline{x_2}}{\sqrt{(\overline{x_1^2} - (\overline{x_1})^2)(\overline{x_2^2} - (\overline{x_2})^2)}},$$

where represents the average of the quantity under it. The figure below plots α_1 (labelled 1, in blue) and α_2 (labeled 2, in red) as a function of r.

Of course, only the region $-1 \leq r \leq +1$ is allowed. Note than in this region α_1 and α_2 have a strong dependence on r. Of course, if there is no collinearity (i.e. $r = 0$) then α_1 and α_2 coincide with a_1 and a_2 (as they should). However, for sufficiently large collinearity α_2 even becomes negative. In this particular example, if $r > 0.4$ - which by the way is a modest amount of collinearity - then α_2 switches sign.



The effect of collinearity on multiple regression coefficients.

Why is this problematic? Recall that a_2 is positive (0.3), which means that an increase in x_2 is accompanied by an increase in y. This behavior would be evident if we plotted the data values for y and x_2 on a scatterplot of y vs. x_2. However, a negative value of α_2 (which occurs if $r > 0.4$) would suggest that an increase in x_2 should be accompanied by a *decrease* in y, which would be the wrong conclusion.

This does not mean that there is a "bug" in regression. In fact, it is entirely possible that the model with both inputs will outperform either of the single-input models. It simply means that the weights cannot be interpreted as reflecting the true relationship between the target and the respective input variables.

I have shown that the weights are meaningless if there is collinearity. But you may ask "Maybe there is no collinearity in my data?" Maybe. But also ask yourself this: what are the chances that a set of variables that you have selected as inputs are collinear? After all, these variables are all supposed to predict the same target. So, it is likely that there is some relationship between the input variables. It's only a question of how linear the relationship is. In my experience a situation where the input variables are not collinear is extremely rare (to nonexistent). There is almost always some amount of collinearity, and it does not take much to render the weights meaningless.

In fact it is good practice to check for collinearity, anyway, because one may be able to exclude some set of the inputs variables from further examination. With less inputs, the number of weights will be less, thereby reducing the chance of overfitting the data.

There are numerous other problems with interpreting regression weights, and the situation is simply exacerbated in nonlinear regression or neural networks. For instance, the existence of hidden nodes robs the inputs of a direct connection to the output that may be interpreted as the predictive strength of that input variable. Or the values of the weights in an NN can change drastically from one local minimum to another.

In short, be very very careful in looking inside the black box to see what it is doing.