Introduction to Neural Networks
U. Minn. Psy 5038

Unifying neural network computations using Bayesian decision theory

Initialize

■ **Read in Statistical Add-in packages:**

```
Off[General::spell1];
<< Statistics`DescriptiveStatistics`
<< Statistics`DataManipulation`
<< Statistics`NormalDistribution`
<< Statistics`MultiDescriptiveStatistics`
<< Statistics`MultinormalDistribution`
<< Statistics`LinearRegression`
<< Graphics`MultipleListPlot`
```

Bayesian Decision Theory

For a recent discussion of Bayesian decision theory in the context of object perception, see:

Kersten, D., Mamassian P & Yuille A (in press) Object perception as Bayesian inference. Annual Review of Psychology.

http://arjournals.annualreviews.org/doi/pdf/10.1146/annurev.psych.55.090902.142005

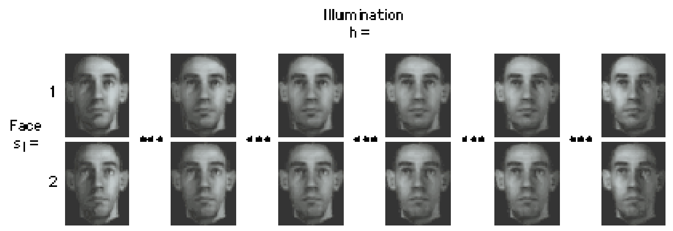
Bayes Decision theory, loss, and risk

Minimizing error isn't always the best thing to do.

The costs of certain kinds of errors (e.g. a high cost to false alarms) could affect the decision criterion. Consider light discrimination. Even though the sensitivity of the observer is essentially unchanged (e.g. the d' for the two Gaussian distributions (bright vs. dim light) remains unchanged), increasing the criterion for deciding bright or dim can increase the overall error rate. This isn't necessarily bad. (d' is the signal-to-noise ratio, given by the difference between the two Gaussian means divided by the standard deviation, which is assumed to be the same.)

A doctor might say that since stress EKG's have about a 30% false alarm rate, it isn't worth doing. The cost of a false alarm is high--at least for the HMO, with the resulting follow-ups, angiograms, etc.. And some increased risk to the patient of extra unnecessary tests. But, of course, false alarm rate isn't the whole story, and one should ask what the hit rate (or alternatively the miss rate) is? Miss rate is about 10%. (Thus, d' is actually pretty high--what is it?). From the patient's point of view, the cost of a miss is very high, one's life. So a patient's goal would not be to minimize errors (i.e. probability of a mis-diagnosis), but rather to minimize a measure of subjective cost that puts a very high cost on a miss, and low cost on a false alarm.

Although decision theory in vision has traditionally been applied to analogous trade-offs that are more cognitive than perceptual, recent work has shown that perception has implicit, unconscious trade-offs in the kinds of errors that are made.



One example is in shape from shading. An image provides the "test measurements" that can be used to estimate an object's shape and/or estimate the direction of illumination. Accurate object identification often depends crucially on an object's shape, and the illumination is a confounding (secondary) variable. This suggests that visual recognition should put a high cost to errors in shape perception, and lower costs on errors in illumination direction estimation. So the process of perceptual inference depends an task. The effect of marginalization in the fruit example illustrated task-dependence. Now we show how marginalization can be generalized through decision theory to model other kinds of goals than error minimization (MAP) in task-dependence.

Bayes Decision theory provides the means to model perceptual/cognitive performance as a function of utility.

Decision theory

Some terminology. We clearly distinguish the decision space from the state or hypothesis space, and introduce the idea of a loss L(d,s), which is the cost for making the decision d, when the actual state is s.

Often we can't directly measure s, and we can only infer it from observations. Thus, given an observation (image measurement) x, we define a risk function that represents the *average loss* over signal states s:

$$R(d; x) = \sum_{s} L(d, s)\, p(s \mid x) \tag{1}$$

This suggests a decision rule: $\alpha(x)=\underset{d}{\operatorname{argmin}}\ R(d;x)$. But not all x are equally likely. This decision rule minimizes the expected risk average over all observations:

$$R(\alpha) = \sum_{s} R(d; x)\, p(x) \tag{2}$$

We won't show them all here, but with suitable choices of likelihood, prior, and loss functions, we can derive standard estimation procedures (maximum likelihood, MAP, estimation of the mean) as special cases.

For the MAP estimator,

$$R(d; x) = \sum_{s} L(d, s)\, p(s \mid x) = \sum_{s} (1 - \delta_{d,s})\, p(s \mid x) = 1 - p(d \mid x) \tag{3}$$

where $\delta_{d,s}$ is the discrete analog to the Dirac delta function--it is zero if d≠s, and one if d=s.

Thus minimizing risk with the loss function L = $(1 - \delta_{d,s})$ is equivalent to maximizing the posterior, p(d|x). And we saw that for a large class of distibutions (exponential), maximizing the posterior is equivalent to minimizing an energy function, as with the Hopfield and Boltzmann networks.

What about marginalization? You can see from the definition of the risk function, that this corresponds to a uniform loss:

L = -1.

$$R(s1; x) = \sum_{s2} L(d2, s2)\, p(s1, s2 \mid x) \tag{4}$$

So for our face recognition example, a really huge error in illumination direction has the same cost as getting it right.

For the fruit example, optimal classification of the fruit identity required marginalizing over fruit color--i.e. effectively treating fruit color identification errors as equally costly

...even tho', doing MAP after marginalization effectively means we are not explicitly identifying color.

Graphical models for Hypothesis Inference: Three types

Three types of nodes in a graphical model: known, unknown to be estimated, unknown to be integrated out (marginalized)

We have three basic states for nodes in a graphical model: known , unknown to be estimated, unknown to be integrated out (marginalization). We have causal state of the world S, that gets mapped to some image data I, through some intermediate parameters M: S->M->I.

So face identity S determines facial shape M, which in turn determines the image I itself. Consider three very broad types of task:

Image synthesis (forward, generative model): We want to model I through p(I|S). In our example, we want to specify "Bill", and then p(I|S="Bill") can be used implemented as an algorithm to spit out images of Bill. M gets integrated out.

Hypothesis inference: we want to model samples for S : p(S|I). Given an image, we want a routine to output likely object identities, so that we can minimize risk, or do MAP classification for accurate object identification. M gets integrated out again.

Learning: we want to model M: p(M|I,S), to learn how the intermediate variables are distributed. Given lots of samples of objects and their images, we want to learn the mapping parameters between them. (Alternatively, do a mental switch and consider a neural network in which an input S gets mapped to an output I through intermediate variables M. We can think of M as representing synaptic weights to be learned.)

Classification: estimating parameters that predict discrete labels. E.g. face identity given an image

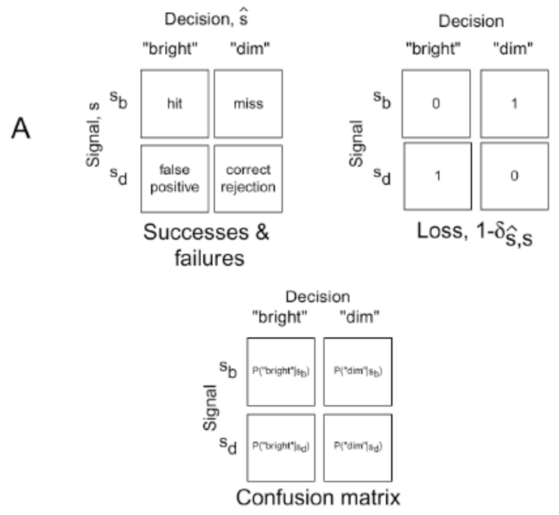
Regression: estimating parameters that provide a good fit to data. E.g. slope and intercept for a straight line through points $\{x_i, y_i\}$.

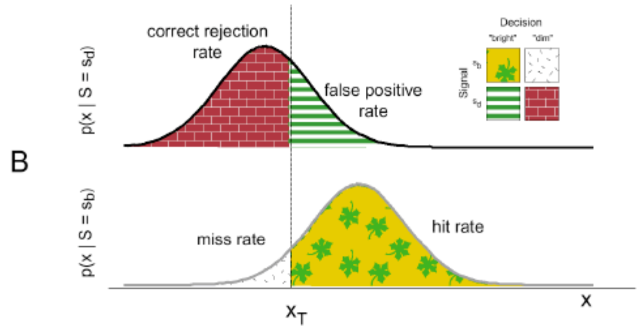
Density estimation: Regression on a probability density functions, with the added condition that the area under the fitted curve must sum to one.

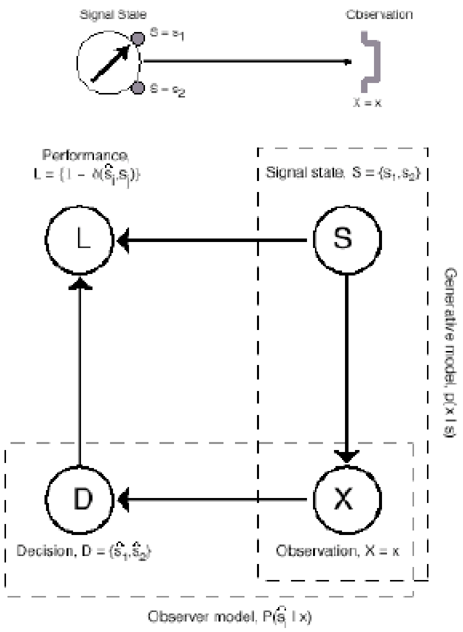
Hypothesis inference: Three types

Detection

Let the decision variable d, be represented by $\hat{s}$.

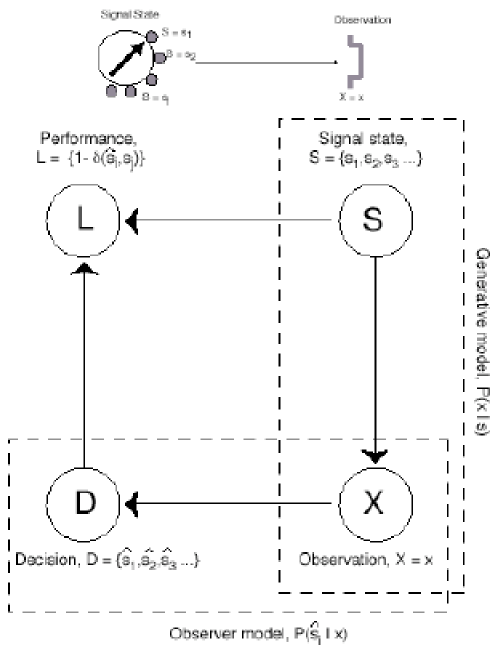





Signal State
S = s_1
S = s_2
Observation
X = x
Performance
L = {1 − $\delta(\hat{s}_i, s_j)$}
Signal state, S = {s_1, s_2}
L
S
D
X
Generative model, p(x | s)
Decision, D = ($\hat{s}_1$, $\hat{s}_2$)
Observation, X = x
Observer model, P($\hat{s}_i$ | x)

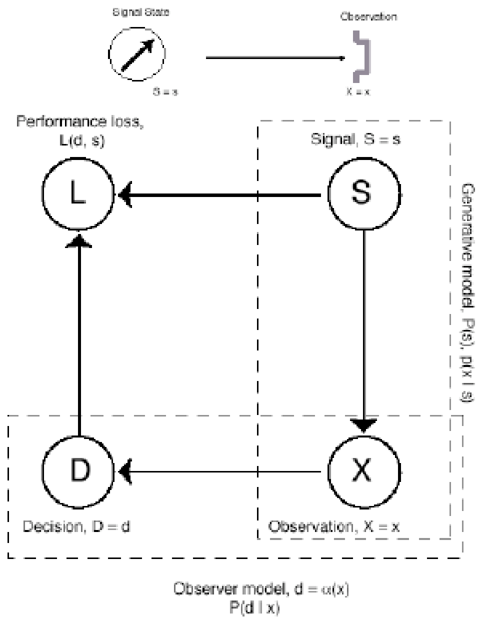
■ Classification

MAP rule: $\underset{i}{\text{argmax}}\{p(S_i \mid x)\}$.



■ **Continous estimation**

$\underset{S}{\text{argmax}}\{p(S \mid x)\}$



As described above, one can show that L(d,s) = -(d-s)^2 produces an estimator that finds the mean, L(d,s) = -δ(d-s), does MAP (i.e. finds the mode), and L(d,s) = 1 is equivalent to marginalization (integrating out s).

Statistical learning, model selection & the bias/variance dilemma

Above we summarized optimal rules for minimizing risk, assuming that we know the distributions of the generative model.

But what if we don't? This is the topic of statistical learning theory.

Use the following link for the notes:

http://gandalf.psych.umn.edu/~kersten/kersten-lab/courses/Psy5038WF2003/MathematicaNotebooks/Lect_27_BiasVariance/biasvarianceNotes.pdf

Consider the regression problem, fitting data that may be a complex function of the input.

The problem in general is how to choose the function that both remembers the relationship between **x** and **y**, and generalizes with new values of **x**. At first one might think that it should be as general as possible to allow all kinds of maps.

For example, if one is fitting a curve, you might wish to use a very high-order polynomial, or a back-prop network with lots of hidden units. There is a draw back to the flexibility afforded by extra degrees of freedom in fitting the data. We can get drastically different fits for different sets of data that are randomly drawn from the same underlying process. The fact that we get different fit parameters (e.g. slope of a regression line) each time means that although we may exactly fit the data each time, we introduce variation between the average fit (over all data sets) and the fit for a single data set. We could get around this problem with a huge amount of data, but the problem is that the amount of required data can grow exponentially with the order of the fit--an example of the so-called "curse of dimensionality".

On the other hand, if the function is restrictive, (e.g. straight lines through the origin), then we will get similar fits for different data sets, because all we have to adjust is one parameter--the slope. The problem here, is that the fit is only good if the underlying process is in fact a straight line through the origin. If it isn't a straight line for instance, there will be a fixed error or **bias** that will never go away, no matter how much data we collect. Statisticians refer to this problem as the *bias/variance* dilemma.

To sum up, lots of parameter flexibility (or lots of hidden units) has the benefit of fitting anything, but at the cost of sensitivity to variability in the data set--there is *variance* introduced by the fits found over multiple training sets (e.g. of a small fixed size).

A fit with very few parameters is not as sensitive to the inevitable variability in the training set, but can give large constant errors or *bias* if the data do not match the underlying model.

There is no general way of getting around this problem, and neural networks are no exception. We generalized linear regression to non-linear fits using error back-propagation. Because back-propagation models can have lots of hidden layers with many units and weights, they form a class of very flexible approximators and can fit almost any function. But these models can show high variability in their fits from one data set to the next, even when the data comes from the same underlying process. Lots of hidden units can mean low bias, but at a high cost in variance.

Demonstration of bias/variance

Warning: Section has not been debugged yet!

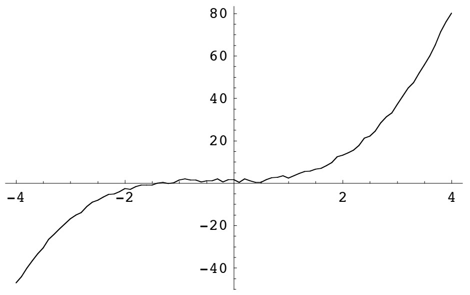
$$\int R(\alpha_D;x)P(\alpha_D)d\alpha = \int (y-\hat{y}(x))^2 P(y|x)dy+\int (\bar{f}(x)-f(x,\alpha))^2$$

■ ***Mathematica*'s regression package**

Go to Help, and find the Linear Regression package. Look up **Regress**. We are going to use Regress as our learning model. We could have used our erro-back prop network, or other learning algorithms that produce a set of fit parameters. The principles would be the same.

```
In[38]:= ff[x_, α_] := α.{1, x, x^2, x^3} + 2 * Random[];

         α = {0, 0, 1, 1};
         xd = Table[{x, ff[x, α]}, {x, -4, 4, .1}];
         ListPlot[xd, AxesOrigin → {0, 0}, PlotJoined → True];
         Regress[xd, {1, x, x^2, x^3}, x][[1]]
```



```
Out[42]=                            Estimate     SE           TStat       PValue
                            1       1.01795      0.0969741    10.4971     0.
         ParameterTable →   x       -0.0470436   0.0691544    -0.680269   0.498375
                            x^2     0.983275     0.0132233    74.359      0.
                            x^3     1.00737      0.00644313   156.348     0.
```

Learning from one data set

■ **True model**

```
In[591]:= ff[x_, α_] := α.{1, x, x^2, x^3};
```

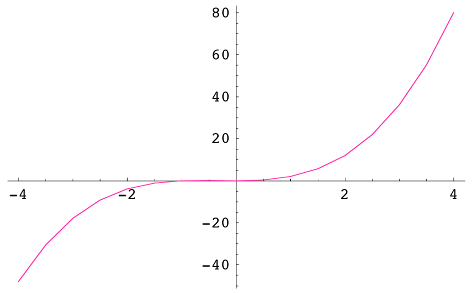
■ **Generative data process: true plus some noise**

```
In[600]:= noise = 15;
          ffn[x_, α_] := ff[x, α] + Random[Real, {-noise, noise}];
```

■ **Choose domain and true model parameters. Calculate true model values ffp, evaluated at xp.**

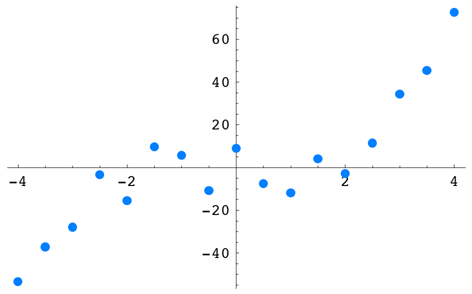
In[602]:=
```
xp = Table[x, {x, -4, 4, .5}];
α = {0, 0, 1, 1};

ffp = ff[#1, α] & /@ xp;
gffp = ListPlot[Transpose[{xp, ffp}],
   PlotStyle → {PointSize[0.02], Hue[.9]}, PlotJoined → True];
```



■ **Run one experiment to collect y values for data process:**

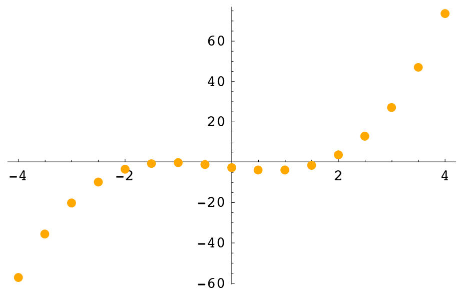
In[606]:=
```
y = ffn[#1, α] & /@ xp;
gy = ListPlot[Transpose[{xp, y}], PlotStyle → {PointSize[0.02], Hue[.6]}];
```



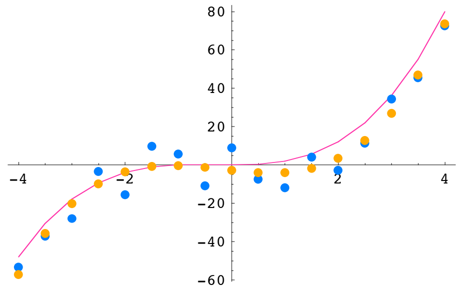
■ **Estimate model parameters using polynomial regression. Return model parameters, and predicted responses, fsquiggle**

In[608]:=
```
αD = Regress[Transpose[{xp, y}], {1, x, x^2, x^3},
   x, RegressionReport -> BestFitParameters][[1, 2]]
fsquiggle = Regress[Transpose[{xp, y}], {1, x, x^2, x^3},
    x, RegressionReport → PredictedResponse][[1, 2]];
gfsquiggle = ListPlot[Transpose[{xp, fsquiggle}],
   PlotStyle → {PointSize[0.02], Hue[.1]}];
```

Out[608]=
```
{-2.85345, -3.04409, 0.686806, 1.21203}
```



In[611]:=
```
Show[gffp, gy, gfsquiggle];
```



Repeat the above, and notice how the model parameters and the fit changes. Try changing the basis functions used for fitting.

Comparing learning from multiple data sets

We'd like some idea of how learning generalizes. So now we are going to run a bunch of experiments to get a bunch of data sets. From these we get multiple fits. Thus we can get estimates of the average value of y, and the variance of the model predictions (with respect to the true model) over multiple fits. And calculate estimates of the second and third terms below.

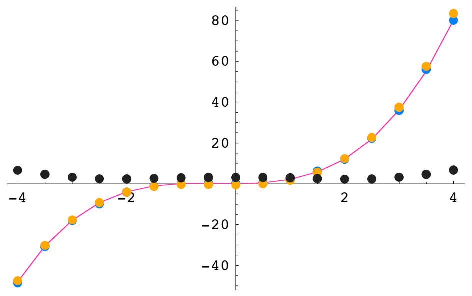
$$\int R(\alpha_D;x)P(\alpha_D)d\alpha = \int (y-\hat{y}(x))^2P(y|x)dy + \int (\tilde{f}(x)-f(x,\alpha))^2P(\alpha_D)d\alpha_D + (\hat{y}$$

■ The right model

First, assume by some incredibly lucky guess, we've chosen the right model {x^2,x^3}, and want to find the parameters.

In[612]:=
```
iter = 500;
ysum = Table[0, {i, 1, Length[xp]}];
fsquigglesum = Table[0, {i, 1, Length[xp]}];
var = Table[0, {i, 1, Length[xp]}];
For[i = 1, i ≤ iter, i++,
  y = ffn[#1, α] & /@ xp;
  fsquiggle = Regress[Transpose[{xp, y}],
     {x^2, x^3}, x, RegressionReport → PredictedResponse][[1, 2]];
  ysum = ysum + y;
  fsquigglesum = fsquigglesum + fsquiggle;
  var = var + (ffp - fsquiggle)^2;
 ];
yhat = ysum / iter;
var = Sqrt[var / iter];
```

In[619]:=
```
gffp = ListPlot[Transpose[{xp, ffp}], PlotStyle → {PointSize[0.02], Hue[.9]},
   PlotJoined → True, DisplayFunction → Identity];
gy = ListPlot[Transpose[{xp, yhat}], PlotStyle → {PointSize[0.02], Hue[.6]},
  DisplayFunction → Identity]; gvar = ListPlot[Transpose[{xp, var}],
  PlotStyle → {PointSize[0.02], GrayLevel[.1]}, DisplayFunction → Identity];
gfsquiggle = ListPlot[Transpose[{xp, fsquiggle}],
  PlotStyle → {PointSize[0.02], Hue[.1]}, DisplayFunction → Identity];
Show[gffp, gy, gfsquiggle, gvar, DisplayFunction → $DisplayFunction];
```



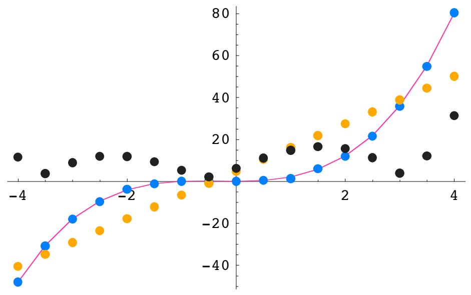
■ A wrong, but simple model

But now suppose that we are trying to fit the data with an inappropriate model. In particular, suppose that it is weak, say a linear model:

```
iter = 500;
ysum = Table[0, {i, 1, Length[xp]}];
fsquigglesum = Table[0, {i, 1, Length[xp]}];
var = Table[0, {i, 1, Length[xp]}];
For[i = 1, i ≤ iter, i++,
  y = ffn[#1, α] & /@ xp;
  fsquiggle = Regress[Transpose[{xp, y}],
     {1, x}, x, RegressionReport → PredictedResponse][[1, 2]];
  ysum = ysum + y;
  fsquigglesum = fsquigglesum + fsquiggle;
  var = var + (ffp - fsquiggle)^2;
 ];
yhat = ysum / iter;
var = Sqrt[var / iter];
```

In[659]:=

```
gffp = ListPlot[Transpose[{xp, ffp}], PlotStyle → {PointSize[0.02], Hue[.9]},
   PlotJoined → True, DisplayFunction → Identity];
gy = ListPlot[Transpose[{xp, yhat}], PlotStyle → {PointSize[0.02], Hue[.6]},
  DisplayFunction → Identity]; gvar = ListPlot[Transpose[{xp, var}],
  PlotStyle → {PointSize[0.02], GrayLevel[.1]}, DisplayFunction → Identity];
gfsquiggle = ListPlot[Transpose[{xp, fsquiggle}],
  PlotStyle → {PointSize[0.02], Hue[.1]}, DisplayFunction → Identity];
Show[gffp, gy, gfsquiggle, gvar, DisplayFunction → $DisplayFunction];
```



The bias is the squared difference between the average y values (blue) and the model fits (orange). The variance is the squared difference between the predicted responses (blue) and the true (red). So we can see that the bias is high. The variance (represented by the square root of the variance, black points in the graph) is not zero. But how does it compare with a model with lots of parameters.

■ Too many parameters

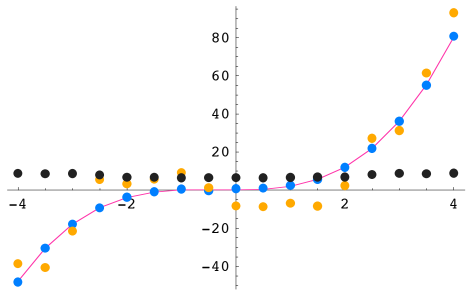
Now let's try over-fitting. (Analogous to having lots of hidden units and/or layers in a non-linear feedforward network).

In[632]:=

```
iter = 500;
ysum = Table[0, {i, 1, Length[xp]}];
fsquigglesum = Table[0, {i, 1, Length[xp]}];
var = Table[0, {i, 1, Length[xp]}];
For[i = 1, i ≤ iter, i++,
  y = ffn[#1, α] & /@ xp;
  fsquiggle = Regress[Transpose[{xp, y}], {1, x, x^2, x^3, x^4, x^5, x^6, x^7, x^8,
      x^9, x^10, x^11, x^12}, x, RegressionReport → PredictedResponse][[1, 2]];
  ysum = ysum + y;
  fsquigglesum = fsquigglesum + fsquiggle;
  var = var + (ffp - fsquiggle)^2;
 ];
yhat = ysum / iter;
var = Sqrt[var / iter];
```

In[639]:=

```
gffp = ListPlot[Transpose[{xp, ffp}], PlotStyle → {PointSize[0.02], Hue[.9]},
   PlotJoined → True, DisplayFunction → Identity];
gy = ListPlot[Transpose[{xp, yhat}], PlotStyle → {PointSize[0.02], Hue[.6]},
  DisplayFunction → Identity]; gvar = ListPlot[Transpose[{xp, var}],
  PlotStyle → {PointSize[0.02], GrayLevel[.1]}, DisplayFunction → Identity];
gfsquiggle = ListPlot[Transpose[{xp, fsquiggle}],
  PlotStyle → {PointSize[0.02], Hue[.1]}, DisplayFunction → Identity];
Show[gffp, gy, gfsquiggle, gvar, DisplayFunction → $DisplayFunction];
```



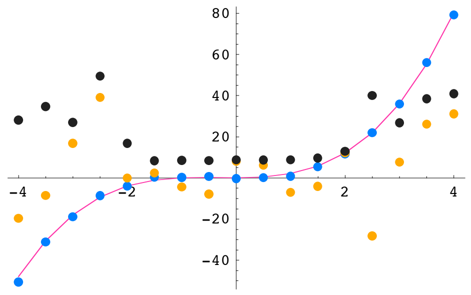
Note how the bias (descrepancy between the orange and blue) is lower than with too few parameters. But we have higher variance than with the "right model" family.

■ Wrong model family and too many parameters

Now try a completely different model class, a subset of basis functions for a Fourier series fit:

```
iter = 500;
ysum = Table[0, {i, 1, Length[xp]}];
fsquigglesum = Table[0, {i, 1, Length[xp]}];
var = Table[0, {i, 1, Length[xp]}];
For[i = 1, i ≤ iter, i++,
  y = ffn[#1, α] & /@ xp;
  fsquiggle = Regress[Transpose[{xp, y}],
     {Sin[x], Cos[x], Sin[2 x], Cos[2 x], Sin[3 x], Cos[3 x], Sin[4 x], Cos[4 x], Sin[5 x],
      Cos[5 x], Sin[6 x], Cos[6 x]}, x, RegressionReport → PredictedResponse][[1, 2]];
  ysum = ysum + y;
  fsquigglesum = fsquigglesum + fsquiggle;
  var  = var + (ffp - fsquiggle)^2;
 ];
yhat = ysum / iter;
var = Sqrt[var / iter];
```

In[649]:=
```
gffp = ListPlot[Transpose[{xp, ffp}], PlotStyle → {PointSize[0.02], Hue[.9]},
   PlotJoined → True, DisplayFunction → Identity];
gy = ListPlot[Transpose[{xp, yhat}], PlotStyle → {PointSize[0.02], Hue[.6]},
  DisplayFunction → Identity]; gvar = ListPlot[Transpose[{xp, var}],
  PlotStyle → {PointSize[0.02], GrayLevel[.1]}, DisplayFunction → Identity];
gfsquiggle = ListPlot[Transpose[{xp, fsquiggle}],
  PlotStyle → {PointSize[0.02], Hue[.1]}, DisplayFunction → Identity];
Show[gffp, gy, gfsquiggle, gvar, DisplayFunction → $DisplayFunction];
```



Appendix

MAP minimizes probability of error: Proof for detection

Here is why MAP minimizes average error. Suppose that x is fixed at a value for which P (S =sb |x)>P (S =sd |x). This is exactly like the problem of guessing “heads ”or “tails ” for a biased coin, say with a probability of heads P (S =sb|x). Imagine the light discrimination experiment repeated many times and you have to decide whether the switch was set to bright or not –but only on those trials for which you measured exactly x .The optimal strategy is to always say “bright ”. Let's see why. First note that:

p(error|x) = p(say "bright", actually dim |x) + p(say "dim", actually bright |x) =

$p(\hat{s1}, s2 \mid x)+p(s1, \hat{s2} \mid x)$

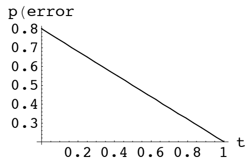
Given x, the response is independent of the actual signal state (see graphical model for detection above--"response is conditionally independent of signal state, given observation x"), so the joint probabilities factor:

p(error|x) = p(say "bright" |x)p(actually dim |x) + p(say "dim" |x)p(actually bright |x)

Let **t= p(say "bright" |x)**, then

p(error|t,x) = t*p(actually dim |x) + (1-t)*p(actually bright |x).

p(error|t), as a function of t, defines a straight line with slope **p(actually dim |x)-p(actually bright |x)**. (Just take the partial derivative with respect to t.) We've assumed P (S =sb |x)>P (S =s d |x)), so p(error|t) has a negative slope, with the smallest non-negative value of t being one. So, error is minimized when t=p(say "bright" |x)=1. I.e. Always say "bright".



Always saying "bright" results in a probability of error P (error |x)=P (S =sd |x).That’s the best that can be done on average. On the other hand, if the observation is in a region for which P (S =sd|x)>P (S =sb |x),the minimum error strategy is to always pick “dim” with a resulting P (error |x)=P (S =sb |x).Of course, x isn’t fixed from trial to trial, so we calculate the total probability of error which is determined by the specific values where signal states and decisions don ’t agree:

$$\text{p(error)}=\sum_{i\neq j} p(\hat{s}_i, s_j)$$
$$=\sum_{i\neq j}\int p(\hat{s}_i, s_j \mid x)\, p(x)\, dx=\sum_{i\neq j}\int p(\hat{s}_i \mid x)\, p(s_j \mid x)\, p(x)\, dx$$

Because the MAP rule ensures that $p(\hat{s}_i, s_j \mid x)$ is the minimum for each x, the integral over all x minimizes the total probability of error.

Nearest neighbor classifier

http://cgm.cs.mcgill.ca/~soss/cs644/projects/simard/

References

Duda, R. O., & Hart, P. E. (1973). Pattern classification and scene analysis . New York.: John Wiley & Sons.

Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern classification* (2nd ed.). New York: Wiley.

(Amazon.com)

Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural networks and the bias/variance dilemma. Neural Computation, 4(1), 1-58.

Kersten, D., & Yuille, A. (2003). Bayesian models of object perception. *Current Opinion in Neurobiology, 13*(2), 1-9.

Kersten, D., Mamassian, P., & Yuille, A. (2004). Object perception as Bayesian Inference. *Annual Review of Psychology, 55*.

MacKay, D. J. C. (1992). Bayesian interpolation. *Neural Computation, 4*(3), 415-447.

Ripley, B. D. (1996). *Pattern Recognition and Neural Networks*. Cambridge, UK: Cambridge University Press.

Vapnik, V. N. (1995). *The nature of statistical learning*. New York: Springer-Verlag.

http://neuron.eng.wayne.edu/software.html