

Introduction to Neural Networks

Bayesian decision theory

Initialize:

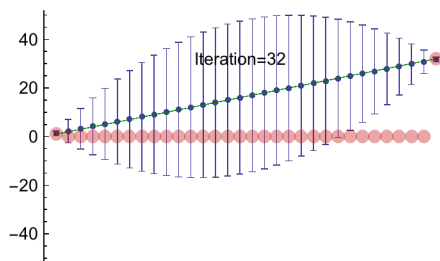
```
Off[General::spell1];  
  
SetOptions[ListDensityPlot, ImageSize -> Small];  
SetOptions[DensityPlot, ImageSize -> Small];  
SetOptions[ContourPlot, ImageSize -> Small];
```

Introduction

Belief propagation

We also computed marginal distributions in the previous lecture's example of belief propagation. Belief propagation is a general technique which can be used to compute marginal distributions and to do MAP estimation. We looked at a very specific example. In particular, our goal was to calculate the marginals:

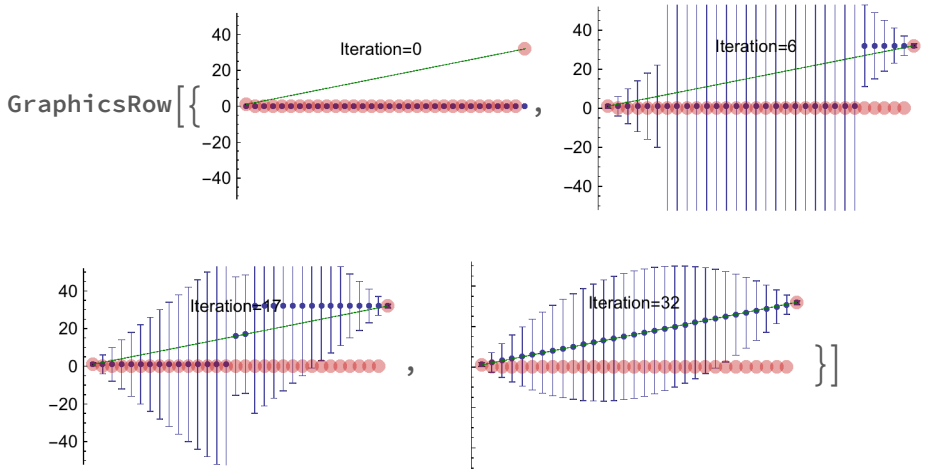
$p(y_i | y_1^*, \dots)$, where y_i represented the depth of an interpolated surface, and $\{y_j^*\}$ represent the measurements of the depth. In our case, the distributions were gaussian. Thus each marginal told us the most probable depth (the mean) and the degree of uncertainty (standard deviation). In our example we had only two data points, one at each end. The rest of the depths were interpolated based on a prior smoothness constraint that encouraged nearby depths to be the same. The result was a "compromise" between what the data "said" and what the prior "thought" answers should usually look like:



This was the final result. To get there, the challenge was to make our best guess of the surface depth given that:

- 1) We didn't have data everywhere. The stereogram was "sparse";
- 2) When we did have depth measurements, they were noisy.

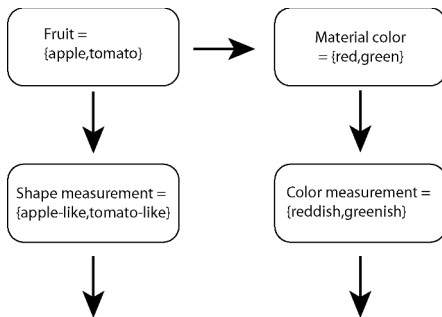
We arrived at the result using belief propagation, which is an iterative method where one updates marginal distributions at each node by receiving and passing messages between neighboring nodes. If there are no loops in the graph, belief propagation provides an exact solution.



Previously...

...a brief review of what is needed today

Natural patterns are complex, and in general it is difficult and often impractical to build a detailed quantitative generative model. But natural inputs, such as sounds and images, do have regularities, and we can get insight into the problem by considering how various factors might produce them. One way to begin simplifying the problem is to note that not all variables have a direct influence on each other. So draw a graph in which lines only connect variables that influence each other. We use directed graphs to represent conditional probabilities.



The the graph specifies how to decompose the joint probability:
 $p[F, C, Is, Ic] = p[Ic | C] p[C | F] p[Is | F] p[F]$

Basic rules: Condition on what is known, integrate out what you don't care about.

Condition on what is known:

Given a state of the world S, and inputs I, the "universe" of possibilities is:

$$p(S, I)$$

If we know I (i.e. the visual system has measured some image feature I), the joint can be turned into a conditional (posterior):

$$p(S | I) = p(S, I) / p(I)$$

Integrate out what we don't care about

Let's split S into: S_{primary} be the variables we care about, $S_{\text{secondary}}$ the ones we don't (also called "noise" or confounding variables or nuisance variables or generic variables).

We don't care to estimate the noise (or other generic, nuisance, or secondary variables):

$$p(S_{\text{primary}} | \mathbf{I}) = \sum_{S_{\text{noise}}} p(S_{\text{primary}}, S_{\text{secondary}} | \mathbf{I}),$$

or if continuous = $\int_{S_{\text{secondary}}} p(S_{\text{primary}}, S_{\text{secondary}} | \mathbf{I}) dS_{\text{secondary}}$

This is the "integrating out" or "marginalization" step. Decisions are then based on the marginals.

Summary of the fruit & color classification example

Pick most probable fruit AND color--

We want to maximize the probability of getting the right fruit and the right color. Both fruit and color are primary variables.

--Answer "red tomato"

Pick most probable color

We want to maximize the probability of getting the right color, and don't care about which fruit it is. Color is primary, fruit type is secondary.

So sum over (marginalize) the fruit type variable to get the marginal $p(\text{color} | \text{measurements})$

--Answer "red"

Pick most probable fruit

We want to maximize the probability of getting the right fruit, and don't care about what color it is. Fruit type is primary, and color is secondary.

So sum over (marginalize) the material color variable to get the marginal $p(\text{fruit} | \text{measurements})$

--Answer "apple"

Today

Generalize the notion of what it means to "care about" a variable. Leads us to Bayesian *decision* theory, where we quantify loss and expected loss (risk).

Show how Bayesian decision theory relates to neural networks and machine learning

Set neural network supervised learning in the context of various statistical/machine learning methods

Bias/variance trade-off in learning

Bayesian Decision Theory: Utility

How to generalize optimal inference to include different degrees of importance in task requirements?

Bayes Decision theory, loss, and risk

We'd now like to generalize the idea of "integrating out" unwanted variables to allow us to put weights on how important a variable is for a task.

Consider a simple discrete decision task in which there is a measurement and one has to decide whether to accept or reject an hypothesis. This is the classic problem of signal detection.

Imagine a noisy image and you have to decide whether it is a human face or not. There are two ways of being correct and two ways of being wrong.

If there really is a face there, and you decide “yes”--that is called a “hit” (or true positive).

If there really is a face there, and you decide “no”, that is a miss (also called a false negative).

If there really is not a face there, and you decide “yes there is a face”--that is called a “false alarm”

(also called a false positive).

If there really is not a face there, and you decide “no”, that is a correct rejection (or true negative).

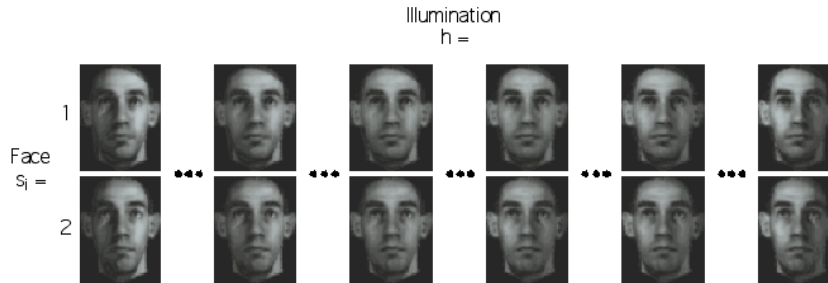


But sometimes getting the benefits of getting the right answer has to be considered in light of the costs of the different types of errors. The costs of certain kinds of errors (e.g. a high cost to false alarms) can affect the decision criterion.

For example, a health professional might say that since stress EKG's have about a 30% false alarm rate, follow-up tests aren't worth doing. The cost of a false alarm is high in dollars, with the resulting follow-ups, angiograms, etc.. And there is 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. If the miss rate is about 10%, from the patient's point of view, the cost of a miss is very high, possibly one's life. So a patient's goal would not be to minimize error rate per se (i.e. probability of a mis-diagnosis, which in effect gives equal weights to both kinds of errors), 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.

Bayes Decision theory, loss, and risk in perception

Although decision theory in vision has traditionally been applied to analogous trade-offs that are more cognitive than perceptual, the concept of utility is relevant in many aspects of perception. Perception has implicit, unconscious trade-offs in the kinds of errors that are made.



For example, image intensities provide the data 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 or nuisance) 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 on 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.

► 1. Why do people often report seeing faces in clouds, tree bark, shower curtains, on mars, in pancakes?

Perhaps it is because the social cost of a false alarm is low compared to the cost of a miss, which is high. If you see a face in a tree trunk, you might be alarmed, but as the saying goes, "better safe than sorry".

Bayes Decision theory provides tools to model performance as a function of utility.

Some terminology. The terms *state*, *hypothesis*, *signal* are essentially the same--to represent the random variable (which could be vector or list) indicating the state of the world--the "state space". We often assume that the decision, d , of the observer maps directly to state space, $d \rightarrow s$. E.g. I estimate that an object is 10 feet away.

We now clearly distinguish the decision space from the state or hypothesis space, and introduce the idea of a loss $L(d, s)$, which is a number representing the cost of making the decision d , when the actual state is s . Loss could be between like variables, such as the cost of estimating a distance of 10 feet when it is actually 11 feet away.

But it could also represent the cost of a decision or action that relies on the true distance. E.g. the cost of choosing an initial velocity (call it d) of a bean bag intended to hit a target 11 feet away.

How to make a decision? As we've seen in human information processing, often we can't directly measure the true value s , and we can only infer it from observations, e.g. a sensory or image measurement x , through a posterior $p(s|x)$.

Thus, given an observation x , we define a risk function that represents the *average loss* over signal states s :

$$R(d; x) = \sum_s L(d, s) p(s|x)$$

This suggests a decision rule, α , that minimizes average loss: $\alpha(x) = \underset{d}{\operatorname{argmin}} R(d; x)$ for that particular x .

But there could be many x 's and not all x are equally likely. This in turn suggests a *decision rule* that minimizes the *expected risk* averaged over all observations:

$$R(\alpha) = \sum_x R(\alpha; x) p(x)$$

Loss functions determine standard inference choices

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) that minimize risk as special cases.

For the MAP estimator,

$$R(d; x) = \sum_s L(d, s) p(s | x) = \sum_s (1 - \delta_{d,s}) p(s | x) = 1 - p(d | x)$$

where as we've seen before, $\delta_{d,s}$ is the discrete analog to the Dirac delta function--it is zero if $d \neq s$, and one if $d = s$. (See KroneckerDelta[])

Thus minimizing risk with the loss function $L = (1 - \delta_{d,s})$ is equivalent to maximizing the posterior, $p(d | x)$. Choosing d that maximizes the posterior effectively penalizes all errors equally.

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

$$L = -1.$$

We don't care how bad (or good) the values of the marginalized variables are, so we give all combinations of d and s the same constant negative loss value $L(d,s) = -1$. So for $L(d_2, s_2) = -1$, minimizing risk is quantitatively equivalent to maximizing the marginal, $p(s_1 | x)$:

$$R(s_1; x) = \sum_{s_2} L(d_2, s_2) p(s_1, s_2 | x) = p(s_1 | x)$$

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

Back again to the fruit color example. With the identical conditional probabilistic structure (e.g. graph) and probabilities, can one ideal decision maker decide "red tomato" and another "apple"? 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.

Note on terminology: Is the glass half full or half empty? Above we described utility theory for pessimists. You may also see terminology favored by optimists, where "utility" or "gain" = - loss, and "expected utility" = - risk.

- 2. Show that quadratic loss, $L(d,s) = (d-s)^2$, is equivalent to estimating the mean

Slant estimation example

This section takes a toy version of "real life" problem, and derives a quantitative prediction of ideal behavior.

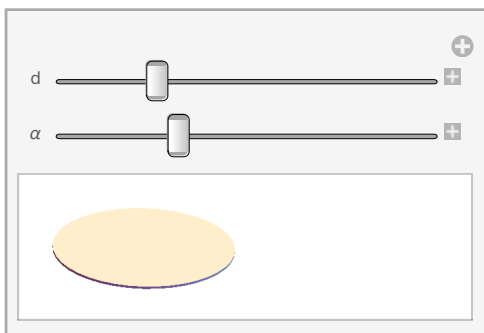
Estimation

Imagine the top of a coffee mug. It typically has a circular cross-section. However, due to projection, the image on your retina is more like an ellipse from most viewpoints. Now imagine it is a "designer coffee mug" which has an elliptical cross-section.

How could you guess the true, i.e. physical 3D shape, from measurements made in the projected image? The "aspect" slider below changes the ratio of the major to minor axes of the coffee mug. The "y" variable changes the slant of your viewpoint. These two causes determine an image measurement x --the height of the projected ellipse in the image (See "Slant" example below).

Manipulate[Graphics3D[

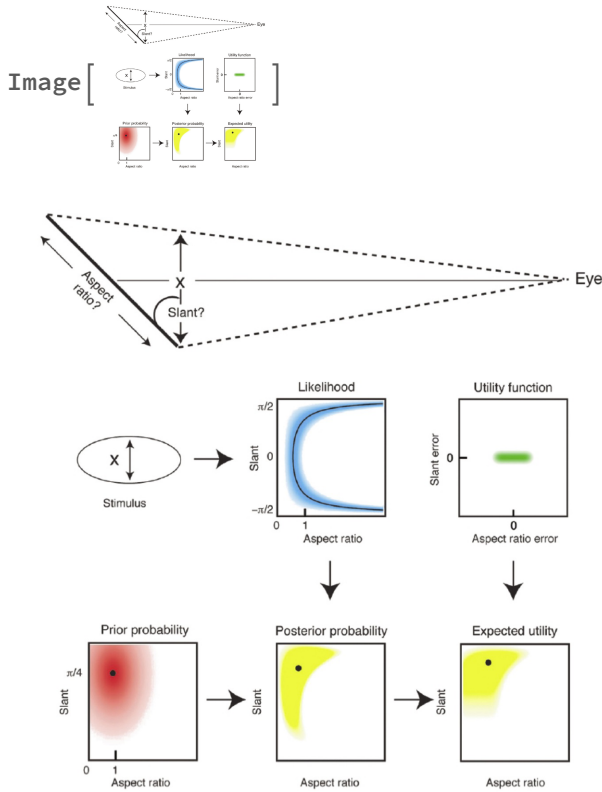
```
{EdgeForm[], Scale[Cylinder[{{-.0, -.01, -.0}, {.0, .01, .0}], 1 / 2], {1, 1, d}],
  Boxed → False, ImageSize → Tiny, ViewCenter → {0, 0, 0},
  ViewPoint → {0, 10, α}], {{d, 1.0}, .1, 2}, {α, -20, 20}]
```



Let x be the data measurement, i.e. the "stimulus". It could be measured by height in the projected image. In our example below, we'll assume that $x = 1/2$.

We approximate the generative model as: $x \approx d \cos[\text{slant}] + \text{noise}$, where d is the physical height of the disk in 3D, and slant is the inclination relative to the viewer. This approximation is reasonable when the disk size is small relative to the viewing distance. We make the simplifying assumption that the width is 1, so then $d = \text{aspect ratio}$.

We'd like to estimate the aspect ratio and the slant. But we have one measurement and two unknowns.



From: Geisler, W. S., & Kersten, D. (2002). Illusions, perception and Bayes. *Nat Neurosci*, 5(6), 508-510.

Introduction

Consider the above figure.

Bayesian ideal observers for tasks involving the perception of objects or events that differ along two physical dimensions, such as aspect ratio and slant, size and distance, or speed and direction of motion. When a stimulus is received, the ideal observer computes the likelihood of receiving that stimulus for each possible pair of dimension values (that is, for each possible interpretation). It then multiplies this likelihood distribution by the prior probability distribution for each pair of values to obtain the posterior probability distribution—the probability of each possible pair of values given the stimulus. Finally, the posterior probability distribution is convolved with a utility function, representing the costs and benefits of different levels of perceptual accuracy, to obtain the expected utility associated with each possible interpretation. The ideal observer picks the interpretation that maximizes the expected utility. (Black dots and curves indicate the maxima in each of the plots.)

As a tutorial example, the figure was constructed with a specific task in mind; namely, determining the aspect ratio (d) of the physical object, and slant (α) of a tilted ellipse. The data is a measurement (x) of the *aspect ratio of the image on the retina*. The black curve in the likelihood plot shows the ridge of maximum likelihood corresponding to the combinations of slant and aspect ratio that are exactly consistent with $x=0.5$; the other non-zero likelihoods occur because of noise in the image and in the measurement of x . The prior probability distribution corresponds to the assumption that surface patches tend to be slanted away at the top and have aspect ratios closer to 1.0.

The asymmetric utility function corresponds to the assumption that it is more important to have an accurate estimate of slant than aspect ratio.

Bayesian decision theory and learning

Bayesian inference seems completely different from the type of neural network feedforward models required to recognize objects. Let's look at how the Bayesian approach relates to alternative models based on neural networks, radial basis functions, or other machine learning methods? Understanding this relationship will also provide a justification for Bayes rule and the intuitions behind it.

Given an image I , we want to classify it as a member of category S . Let a decision rule to estimate an output S be: $S^* = \alpha(I)$ and let the loss function (or negative utility) be $L(\alpha(I), S)$. As discussed above, the loss function is the penalty for making the decision $\alpha(I)$ when the true state is S (e.g., a fixed penalty for a misclassification).

Suppose we have a set of examples $\{I_i, S_i; i=1, \dots, N\}$, then the empirical risk (Vapnik 1998) of the rule $\alpha(I)$ is defined to be:

$$R_{emp}(\alpha) = (1/N) \sum_{i=1}^N L(\alpha(I_i), S_i) \quad (1)$$

For example, empirical risk could be the proportions of misclassifications.

The best decision rule $\alpha^*(\cdot)$ is selected to minimize $R_{emp}(\alpha)$. For example, the decision rule is chosen to minimize the number of misclassifications. Neural networks and machine learning models select rules to minimize various forms of $R_{emp}(\alpha)$.

Now suppose that the samples $\{S_i, I_i\}$ come from a distribution $p(S, I)$ over the set of training pairs. Then, if we have a sufficient number of samples, we can replace the empirical risk by the true risk:

$$R(\alpha) = \sum_I \sum_S L(\alpha(I), S) p(S, I) \quad (2)$$

Note this is just working backwards from what it means to take an average.

Minimizing $R(\alpha)$ leads to a decision rule that depends on the posterior distribution $p(S|I)$ obtained by the product rule. To see this, we rewrite Equation 2 as

$$R(\alpha) = \sum_I p(I) \{ \sum_S p(S|I) L(\alpha(I), S) \}$$

where we have expressed $p(S, I) = p(S|I)p(I)$. So the best decision $\alpha(I)$ for a specific image I is given by

$$\alpha^*(I) = \arg \min_{\alpha} \sum_S p(S|I) L(\alpha(I), S)$$

and depends on the posterior distribution $p(S|I)$ and by Bayes rule, $p(S|I) = p(I|S)p(S)/p(I)$. Hence, Bayes arises naturally when you start from the risk function specified by Equation 2.

There are two take-home messages:

1) the Bayes posterior $p(S|I)$ follows logically from trying to minimize empirical risk, e.g. the number of misclassifications in the empirical risk (provided there are a sufficient number of samples).

2) it is possible to have an algorithm, or a network, that computes $\alpha(\cdot)$ and minimizes the Bayes risk but that does not explicitly represent the probability distributions $p(I|S)$ and $p(S)$.

Graphical model for decision theory: A summary

This section shows the common structure shared by three types of inference: detection (“is the signal there or not?”), classification (“which signal is it?”), and estimation (“what is the quantity?”).

- Detection: $a = s'$, $s' \in \{s_1, \text{not } s'_1\}$
- Classification: $a = s' \in \{s_1, s_2, s_3, s_4 \dots\}$
- Estimation: $a = s'$, where s' takes on continuous values

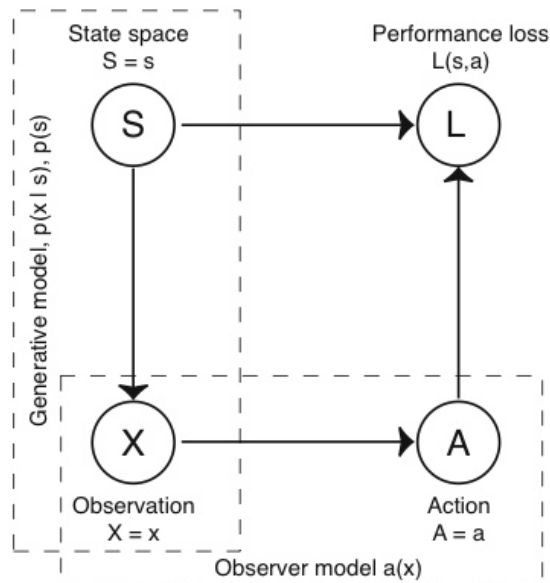
Decisions can be right or wrong regarding a discrete hypothesis (detection, classification), or have some metric distance from an hypothesis along a continuous dimensions (estimation). Each decision or estimation has an associated loss function. There is a common graphical structure to each type of inference.

In the diagram below, we replace the decision variable \mathbf{d} , by a more general term \mathbf{a} for "action".

The observer model might refer to the input/output model of a human, animal, neural population, or neuron. Or it can refer to an ideal observer or ideal agent that minimizes the average loss. An ideal agent is called a “normative” model of a process or behavior...i.e. given a description of the problem to be solved, what is best solution? Then one asks how an human, animal, neural population, or neuron compares.

Decisions, tasks, and actions

The variables in decision theory have a graphical structure (see Kersten and Mamassian, 2009) which determines how to factorize the joint distribution over them, and thus determines a common starting point for deriving ideal observers--i.e. decision makers that minimize risk. The graph below formally summarizes an observer or "agent".



How is utility learned and represented by the brain?

Like Bayesian inference theory, Bayesian decision theory is very general. It provides tools to model behavior, but doesn't specify representations or algorithms. So how is utility represented in the brain? Or learned?

Natural loss functions may be "hard-wired", embedded in the architecture. But also adaptive changes through learning. The role of reward.

The bias/variance dilemma

The problem we now consider is how to choose the function that both remembers the relationship between \mathbf{x} and \mathbf{y} , and generalizes given new values of \mathbf{x} . At first one might think that it should be as general as possible to allow all kinds of mappings.

For example, if one is fitting a apparently complicated curve, you might wish to use a very high-order polynomial, or a back-prop network with lots of hidden units. There is a drawback, however, 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 fits over the ensemble of data sets.

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 the trade-off between simple but biased fits, and complex but data-dependent variation, 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--variability in the model fits given new sets of data.

Demonstration of bias/variance for regression

Suppose we have an unknown, underlying generative model given by: $p(y|x)$ and $p(x)$. From this we obtain a set of samples $\{x_i, y_i\}$, $i=1 \dots N$.

Keep in mind that a set of samples is not in general representative of the whole distribution $p(x,y)$. Later samples may suggest a different model.

We posit some estimator to fit the data: $y_i \sim f(x_i; \theta)$. This function f has some associated parameters, θ , (or neural weights) that need to be estimated. And these parameters get re-estimated each time with get a new set of samples. So there is variation in f .

In general, there will be some cost assigned to errors in f 's ability to predict the y 's. E.g. the expected value of the squared difference between the fits and the true expected value of y , call it \hat{y} . It is insightful to write this cost as the sum of two terms:

$$E[(f - \hat{y})^2] = (E[f] - \hat{y})^2 + E[(f - E[f])^2]$$

The first term on the right is the "bias" (squared), and the second term the "variance". The bias is the "constant error" which tells us how far off our estimator will be from the true value of \hat{y} -- could be non-zero even with an unlimited supply of data. The second term, the "variance of the fits" (e.g over multiple datasets), tells us how much variation we have in the ensemble of fits f about the average of all the fit values, $E[f]$.

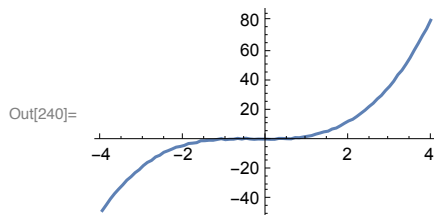
The left hand side of the above equation is similar to the how we formulated loss in decision theory, i.e. a measure of the cost of "deciding f , when true value is \hat{y} ".

Let's demonstrate the effects of the bias/variance trade-off given a generative model.

Mathematica's regression package

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

```
In[237]:= ff[x_, α_] := α.{1, x, x^2, x^3} + RandomReal[];
α = {0, 0, 1, 1};
xd = Table[{x, ff[x, α]}, {x, -4, 4, 0.1}];
ListPlot[xd, AxesOrigin -> {0, 0}, Joined -> True, ImageSize -> Small]
lm = LinearModelFit[xd, {1, x, x^2, x^3}, x];
lm[{"ParameterTable", "RSquared"}]
```



Out[242]=

	Estimate	Standard Error	t-Statistic	P-Value
1	0.485797	0.0421671	11.5208	1.99362×10^{-18}
x	-0.0533437	0.0300703	-1.77396	0.0800226
x ²	1.00015	0.00574989	173.942	1.08147×10^{-101}
x ³	1.00475	0.00280166	358.627	7.40975×10^{-126}

, 0.999907

Learning from one data set

Underlying model space

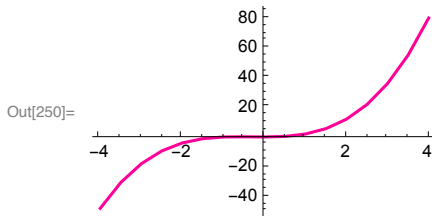
```
In[243]:= Clear[ff];
ff[x_, α_] := α.{1, x, x^2, x^3};
```

Generative data process: true plus some noise

```
In[245]:= noise = 15;
ffn[x_, α_] := ff[x, α] + 1.5 * RandomReal[{-noise, noise}];
```

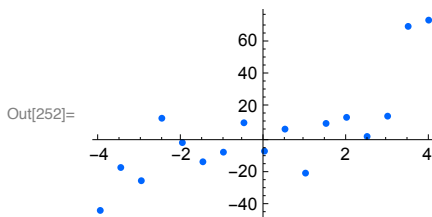
Choose domain and true model parameters. Calculate a set of samples from a true (noise free) model f_p , evaluated at x_p .

```
In[247]:= xp = Table[x, {x, -4, 4, 0.5}];
α = {0, 0, 1, 1};
ffp = (ff[#1, α] &) /@ xp;
gffp = ListPlot[Transpose[{xp, ffp}],
  PlotStyle → {PointSize[0.02], Hue[0.9]}, Joined → True, ImageSize → Small]
```



Run one experiment to collect y values for data process:

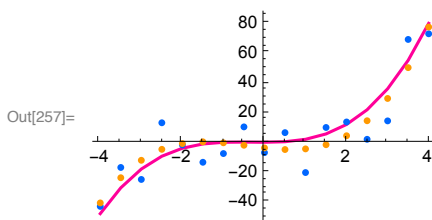
```
In[251]:= y = (ffn[#1, α] &) /@ xp;
gy = ListPlot[Transpose[{xp, y}],
  PlotStyle → {PointSize[0.02], Hue[0.6]}, ImageSize → Small]
```



Estimate model parameters using polynomial regression. Return model parameters, and predicted responses, $f_{sqiggle}$

```
In[253]:= xd = Table[{x, ff[x, α]}, {x, -4, 4, 0.1}];
In[254]:= αD = LinearModelFit[Transpose[{xp, y}], {1, x, x^2, x^3}, x];
fqsquiggle = Table[{x, αD[x]}, {x, -4, 4, 0.5}];
gfsquiggle =
  ListPlot[fqsquiggle, PlotStyle → {PointSize[0.02], Hue[0.1]}, ImageSize → Small];
```

```
In[257]:= 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.

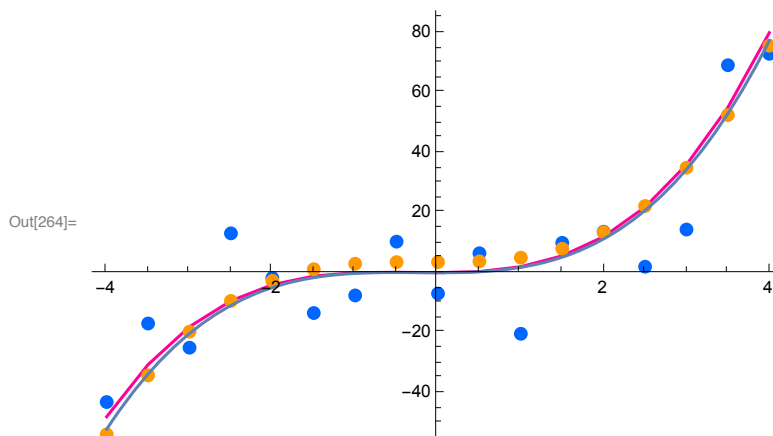
Demonstrating the key idea

We'd like some idea of how learning generalizes depending on model complexity

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[258]:= y = ffn[#1,  $\alpha$ ] & /@ xp;
In[259]:=  $\alpha$ D2 = LinearModelFit[Transpose[{xp, y}], {x^2, x^3}, x];
fsquiggle = Table[{x,  $\alpha$ D2[x]}, {x, -4, 4, 0.5}];
gfsquiggle =
  ListPlot[fsquiggle, PlotStyle -> {PointSize[0.02], Hue[0.1]}, ImageSize -> Small];
In[262]:= gffp = ListPlot[Transpose[{xp, ffp}],
  PlotStyle -> {PointSize[0.02], Hue[0.9]}, Joined -> True];
gsmooth = Plot[Fit[Transpose[{xp, y}], {x^2, x^3}, x] /. x -> x2, {x2, -4, 4}];
Show[gffp, gy, gfsquiggle, gsmooth]
```



A simple, but wrong model

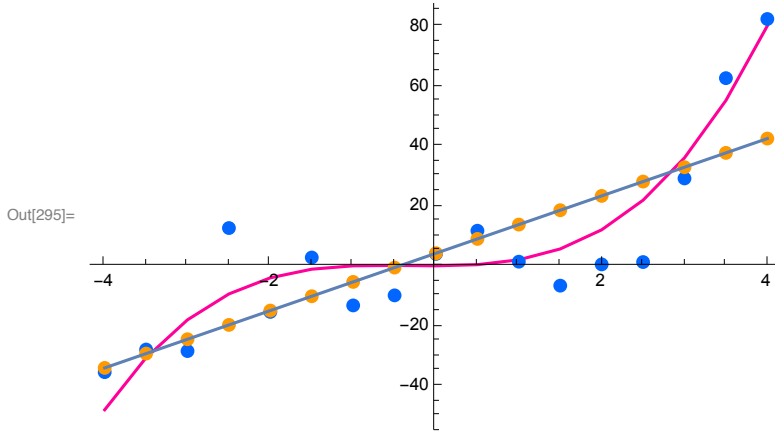
But now suppose that we are trying to fit the data with a wrong, but simple model with few parameters, say a linear model:

```
In[288]:= y = ffn[#1,  $\alpha$ ] & /@ xp;
 $\alpha$ D3 = LinearModelFit[Transpose[{xp, y}], {1, x}, x];
fsquiggle = Table[{x,  $\alpha$ D3[x]}, {x, -4, 4, 0.5}];
gfsquiggle =
  ListPlot[fsquiggle, PlotStyle -> {PointSize[0.02], Hue[0.1]}, ImageSize -> Small];
```

```

In[292]:= gffp = ListPlot[Transpose[{xp, ffp}],
           PlotStyle -> {PointSize[0.02], Hue[0.9]}, Joined -> True];
           gy = ListPlot[Transpose[{xp, y}], PlotStyle -> {PointSize[0.02], Hue[0.6]}];
           gsmooth = Plot[Fit[Transpose[{xp, y}], {1, x}, x] /. x -> x2, {x2, -4, 4}];
           Show[gffp, gy, gfsquiggle, gsmooth]

```



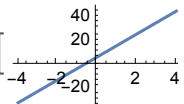
Out[295]=

$$E[(f - \hat{y})^2] = (E[f] - \hat{y})^2 + E[(f - E[f])^2]$$

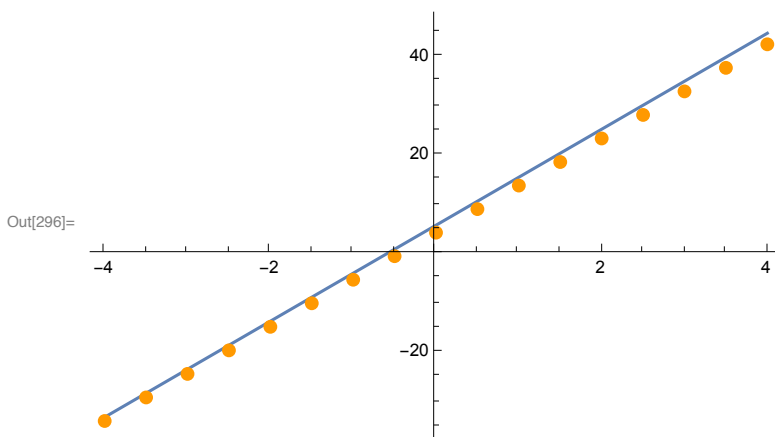
In[274]=

The bias is the squared difference between the expectation of the y values, \hat{y} (red, representing the true values) and the expectation of the model fits (one fit, f , is shown in orange). You can guess that even with more data, the average will still be a straight line, with significant constant error. The variance is based on a measure of the squared difference between the predicted responses, f , (orange) and the average straight line fit (shown below in blue). With many fits, the difference between straight line fits and the true average straight line fit will tend to be small. Below is shown the expected straight line fit (which can be estimated with lots of samples) in blue, together with the particular predictions (a straight line fit) from from the small sample in orange.

In[296]= Show[



, gfsquiggle]



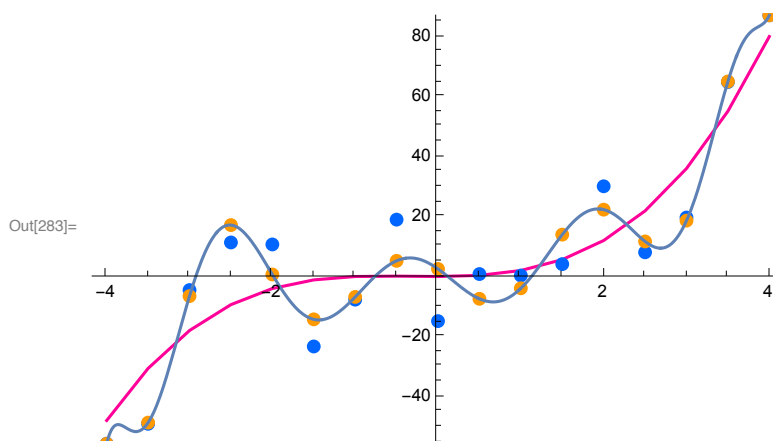
How do bias and variance compare with a model with lots of parameters?

A complex model, with too many parameters

Now let's try over-fitting. This is analogous to having too many hidden units and/or layers in a non-linear feedforward network.

```
In[276]:= y = ffn[#1, α] & /@ xp;
αD4 = LinearModelFit[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];
fsquiggle = Table[{x, αD4[x]}, {x, -4, 4, 0.5}];
gfsquiggle =
  ListPlot[fsquiggle, PlotStyle → {PointSize[0.02], Hue[0.1]}, ImageSize → Small];

In[280]:= gffp = ListPlot[Transpose[{xp, ffp}],
  PlotStyle → {PointSize[0.02], Hue[0.9]}, Joined → True];
gsmooth = Plot[Fit[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] /. x → x2, {x2, -4, 4}];
gy = ListPlot[Transpose[{xp, y}], PlotStyle → {PointSize[0.02], Hue[0.6]}];
Show[gffp, gy, gfsquiggle, gsmooth]
```



Note how the discrepancy between the orange (predicted values) and blue (sampled values) is low—we can achieve this with the large number of parameters. Further, if we did multiple fits, there would be variation about the red line, but on average the discrepancy between orange would be small—i.e. the bias term would be small. However, this would be at the cost of higher variability in the set of fits, f , about the expected value of f (which would be close to the true value). With a limited amount of data, we could have a model that we can't trust to generalize.

Try comparing learning over several data sets to observe the variance in the fits.

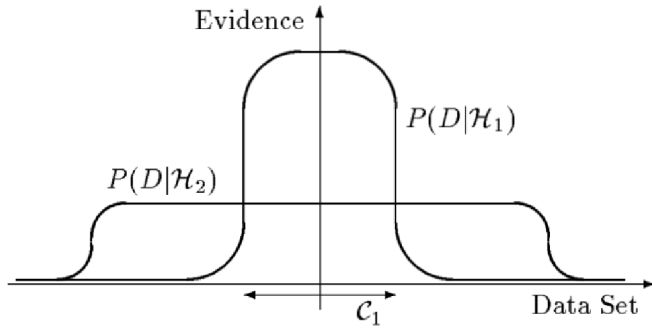
$$E[(f - \hat{y})^2] = (E[f] - \hat{y})^2 + E[(f - E[f])^2]$$

In[236]=

Dealing with over-fitting

Bayesian model selection

MacKay (1992)



$$p(\theta | \text{data}, \text{model}) = \frac{p(\text{data} | \theta, \text{model}) p(\theta | \text{model})}{p(\text{data} | \text{model})}$$

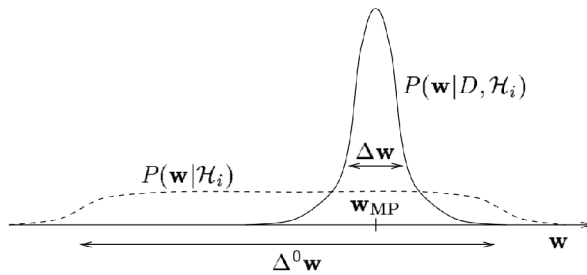
$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{evidence}}$$

$$p(\text{data} | \text{model}) = \int p(\text{data} | \theta, \text{model}) p(\theta | \text{model}) d\theta$$

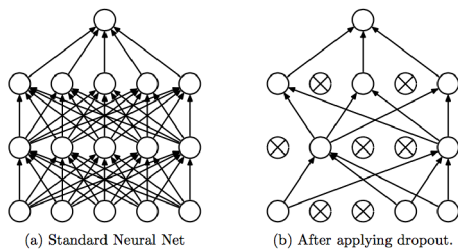
$$p(\text{data} | \text{model}) \approx p(\text{data} | \theta_{\text{mp}}) \times p(\theta_{\text{mp}} | \text{model}) \Delta\theta$$

evidence \approx best fit likelihood \times "Occam factor"

In the figure, $w = \theta$, $w_{\text{mp}} = \theta_{\text{mp}}$, $D = \text{data}$, and $H_i = \text{model}$.



Drop-out & error backpropagation in multi-layer networks



See: <http://arxiv.org/pdf/1207.0580v1.pdf>

and: Srivastava, N., Hinton, G. E., & Krizhevsky, A. (2014). Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15, 1929-1958

Later we'll talk about inference in hierarchical structured graphs.

Cross-validation

Appendix

Mathematica code to illustrate Bayesian estimation of surface slant and aspect ratio

This code was used to produce the figure in a Nature Neuroscience News & Views article by Geisler and Kersten (2002). (See paper by Weiss, Simoncelli and Adelson.)

Initialization

```
npoints = 128;
loaspect = 0;
hiaspect = 5;
$TextStyle = {FontFamily -> "Helvetica", FontSize -> 14}
Fswitch = True;
{FontFamily -> Helvetica, FontSize -> 14}

PadMatrix[mat_, gray_, n_] := Module[{d},
  d = Dimensions[mat];
  Return[PadRight[PadLeft[mat, {d[[1]] + n, d[[2]] + n}, gray],
    {d[[1]] + 2 * n, d[[2]] + 2 * n}, gray]];
];
```

Init delta

```
gdelta[x_, w_] := 1 - (UnitStep[x + w / 2] - UnitStep[x - w / 2]);
(*Plot[gdelta[x,1], {x, -10, 10}, PlotRange -> {0, 2}];*)
```

Calculate Likelihood function and its maxima

$$p(I | S_{prim}, S_{sec})$$

$$p(x | \alpha, d) = p(x - \phi(\alpha, d))$$

$$x = \phi(\alpha, d) + noise$$

(We've used the notion "prim" and "sec" for primary and secondary variables. But below rather than integrating out the secondary variable, we use a loss function to soften the notion of what is important and what is not. We'll require more precision of the slant estimate than of the aspect ratio.)

Image model determines the constraint, $x = d \cos[\alpha] + noise$, determines the likelihood

Assume noise has a Gaussian distribution with standard deviation = 1/5;
 Assume an image measurement (x=1/2)

```
likeli[alpha_, x_, d_, s_] :=
  Exp[-((x - d Cos[alpha])^2) / (2 s^2)] (1 / Sqrt[2 Pi s^2])
likeli[alpha, x, d, s]
x = 1/2; s = 1/5;
like = likeli[alpha, x, d, s]
```

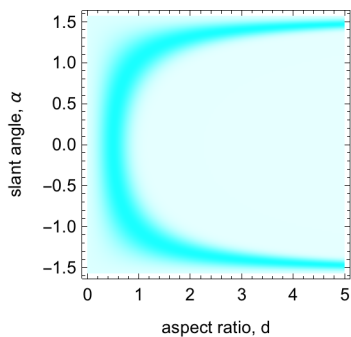
$$\frac{e^{-\frac{(x-d \cos[\alpha])^2}{2s^2}}}{\sqrt{2\pi} \sqrt{s^2}}$$

$$5 e^{-\frac{25}{2} \left(\frac{1}{2} - d \cos[\alpha]\right)^2}$$

$$\frac{5 e^{-\frac{25}{2} \left(\frac{1}{2} - d \cos[\alpha]\right)^2}}{\sqrt{2\pi}}$$

Plot likelihood

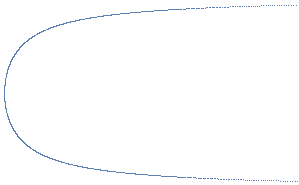
```
gdlike = DensityPlot[like, {d, loaspect, hiaspect}, {alpha, -Pi/2, Pi/2}, PlotPoints -> npoints,
  Mesh -> False, ColorFunction -> (RGBColor[1 - (0.1 + 0.8 #1), 1, 1] &),
  FrameLabel -> {"aspect ratio, d", "slant angle, alpha"}]
```



Plot likelihood maxima

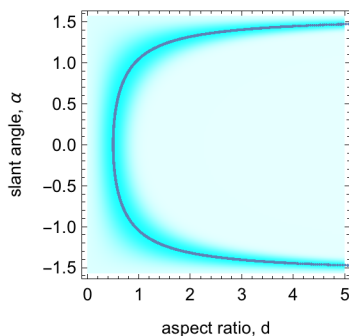
There is no unique maximum. The likelihood function has a ridge

```
gtemp29 = ListPlot[Table[{ $\frac{N[x]}{\text{Cos}[\alpha]}$ , alpha}, {alpha,  $-\frac{\pi}{2}$ ,  $\frac{\pi}{2}$ , 0.001}],
  ImageSize → Small, Axes → False]
```



Plot likelihood together with maximum along the ridge

```
gdlike = DensityPlot[like, {d, loaspect, hiaspect}, { $\alpha$ ,  $-\frac{\pi}{2}$ ,  $\frac{\pi}{2}$ }, PlotPoints → npoints,
  Mesh → False, ColorFunction → (RGBColor[1 - (0.1` + 0.8` #1), 1, 1] &),
  FrameLabel → {"aspect ratio, d", "slant angle,  $\alpha$ "}, Frame → Fswitch];
glikemax = Show[gdlike, gtemp29]
```



Calculate the prior, and find its maximum

$$p(S_{prim}, S_{sec})$$

$$p(\alpha, d)$$

The prior probability distribution corresponds to the assumption that surface patches tend to be slanted away at the top and have aspect ratios closer to 1.0. We model the prior by a bivariate gaussian:

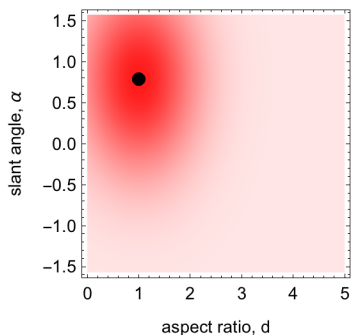
```
PDF[MultinormalDistribution[{ $\mu_\alpha$ ,  $\mu_d$ }, R], { $\alpha$ , d}];
```

```

R1 = {{.25, 0}, {0, .25}};
ndist3 = MultinormalDistribution[{Pi / 4., 1}, R1];
pdf3 = PDF[ndist3, {α, d}];
FindMinimum[-pdf3, {{d, 0}, {α, 1}}]
gdprior = DensityPlot[pdf3^.4, {d, loaspect, hiaspect},
  {α, -Pi / 2, Pi / 2}, PlotPoints → npoints, Mesh → False,
  ColorFunction → (RGBColor[1, 1 - (0.1 + 0.8 #), 1 - (0.1 + 0.8 #)] &),
  FrameLabel → {"aspect ratio, d", "slant angle, α"}];
{-0.63662, {d → 1., α → 0.785398}}

```

```
Show[gdprior, Graphics[{PointSize[0.05`], Point[{1, 0.785`}]}]]
```



Calculate the posterior, and find its maximum

$$p(S_{prim}, S_{sec} | I) \propto p(I | S_{prim}, S_{sec})p(S_{prim}, S_{sec})$$

$$p(\alpha, d | x) = \frac{p(x | \alpha, d)p(\alpha, d)}{p(x)}$$

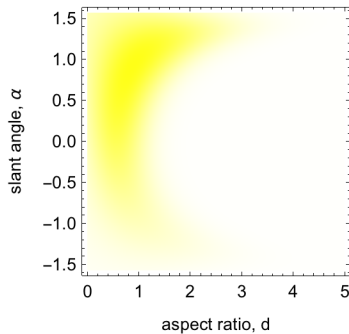
$$p(\alpha, d | x) \propto p(x | \alpha, d)p(\alpha, d)$$

More precisely, we'll calculate a quantity proportional to the posterior. The posterior is equal to the product of the likelihood and the prior, divided by the probability of the image measurement, x . Because the image measurement is fixed, we only need to calculate the product of the likelihood and the prior:

```
Clear[α, x, d, s];
```

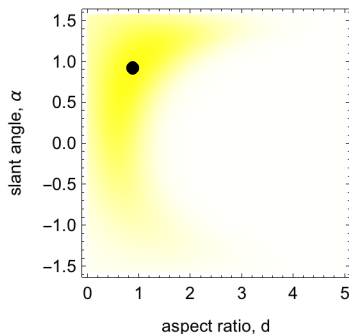
```
likeli[α, x, d, s] * PDF[MultinormalDistribution[{μα, μd}, R], {α, d}];
```

```
gdpost = DensityPlot[(pdf3 * like) ^ .2, {d, loaspect, hiaspect}, {α, -Pi / 2, Pi / 2},
  ColorFunction -> (RGBColor[1, 1, 1 - (0.01 + 0.9 #)] &), PlotPoints -> npoints,
  Mesh -> False, FrameLabel -> {"aspect ratio, d", "slant angle, α"}, Frame -> Fswitch]
```



```
R1 = {{.25, 0}, {0, .25}};
ndist3 = MultinormalDistribution[Pi / 4., 1], R1];
pdf3 = PDF[ndist3, {α, d}]
FindMinimum[-pdf3 * like, {{d, 1}, {α, 1}}]
0.63662 e1/2 (-0.+4. (-1+d)) (-1+d) - (0.+4. (-0.785398+α)) (-0.785398+α)
{-1.17378, {d -> 0.881475, α -> 0.923647}}
```

```
Show[gdpost, Graphics[{PointSize[0.05], Point[{0.88, 0.92}]}]]
```



Compute expected loss--i.e. risk, and find its minimum

The expected loss is given by the convolution of the loss with the posterior:

risk=posterior*loss, where * means convolve; utility= - risk

Loss function

$$l(\Delta\alpha, \Delta d) = l(\alpha' - \alpha, d' - d)$$

The asymmetric utility function corresponds to the assumption that it is more important to have an accurate estimate of slant than aspect ratio. The loss function reflects the task. Accurate estimates of slant may be more important for an action such as stepping, whereas an accurate estimation of aspect

ratio may be more important for determining object shape (circular coffee mug top or not?). If one were to grasp an object that is elliptical in shape with say, the thumb on the bottom and finger on top, this task could require accurate estimates of both slant and aspect ratio.

```
maploss = Table[(1 - gdelta[x1d, 0.25`]) (1 - gdelta[x2d, 2]),
  {x1d, -3, 3,  $\frac{6}{\text{npoints}}$ }, {x2d, -3, 3,  $\frac{6}{\text{npoints}}$ }]];
gdloss = ListDensityPlot[maploss, Mesh → False,
  ColorFunction → (RGBColor[1 - (0.01 + 0.9 #1), 1 - (0.01 + 0.9 #1), 1] &), Frame → False]
```



Convolve posterior with loss function

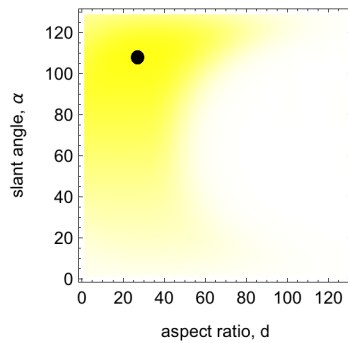
$$utility(\alpha', d') = - \sum_{\alpha, d} p(x | \alpha, d) p(\alpha, d) l(\alpha' - \alpha, d' - d)$$

Convert function description to numerical arrays for convolving

```
post = Transpose[Table[Like * pdf3, {d, loaspect, hiaspect,
  (hiaspect - loaspect) / npoints}, {α, -Pi / 2, Pi / 2, Pi / npoints}]];
post2 = PadMatrix[post, 0, 16];
maploss2 = PadMatrix[maploss, 0, 16];
offset = Floor[Dimensions[maploss2][[1]] / 2];
tempcon = ListConvolve[maploss2, post2, {-1, -1}];
risk2 = RotateLeft[tempcon, {offset, offset}];
risk =
  Take[risk2, {17, Dimensions[risk2][[1]] - 16}, {17, Dimensions[risk2][[1]] - 16}];
grbrisk = ListDensityPlot[Map[#^1. &, risk]^2,
  Mesh → False, ColorFunction → (RGBColor[1, 1, 1 - (0.01 + 0.9 #)] &),
  FrameLabel → {"aspect ratio, d", "slant angle, α"}, Frame → Fswitch];
Position[(risk), Max[(risk)]]
{{108, 27}}
```



```
Show[grbrisk, Graphics[PointSize[0.05], Point[{27, 108}]]]
```



References

- Duda, R. O., Hart, P. E., & Stork, D. G. (2001). *Pattern classification* (2nd ed.). New York: Wiley. (Amazon.com)
- Geisler Wilson S. and Daniel Kersten (2002) Illusions, perception and Bayes. *Nature Neuroscience*, 5 (6), 508-510. Or (pdf).
<http://gandalf.psych.umn.edu/~kersten/kersten-lab/papers/GeislerKerstennn0602-508.pdf>
http://www.ncbi.nlm.nih.gov/entrez/query.fcgi?cmd=Retrieve&db=PubMed&dopt=Citation&list_uids=12037517
- Green, D. M., & Swets, J. A. (1974). *Signal Detection Theory and Psychophysics*. Huntington, New York: Robert E. Krieger Publishing Company.
- Hsu, M., Bhatt, M., Adolphs, R., Tranel, D., & Camerer, C. F. (2005). Neural systems responding to degrees of uncertainty in human decision-making. *Science*, 310(5754), 1680-1683.
- Kersten, D & Mamassian, P (2009) Ideal Observer Theory. In: Squire LR (ed.) *Encyclopedia of Neuroscience*, volume 5, pp. 89-95. Oxford: Academic Press.
- Kersten, Masmassian, P., & Yuille, A. (2004). Object perception as Bayesian inference. *Annual Review of Psychology*, 55, 271–304.
- 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.
- Pillow, J. (2007). Likelihood-based approaches to modeling the neural code. In K. Doya, S. Ishii, A. Pouget, & R. Rao, *Bayesian Brain: Probabilistic Approaches to Neural Coding* (pp. 53–70). MIT Press Cambridge, MA.
- Vapnik, V. N. (1995). *The nature of statistical learning*. New York: Springer-Verlag.
<http://neuron.eng.wayne.edu/software.html>
- Yuille, A. L., & Bülthoff, H. H. (1996). Bayesian decision theory and psychophysics. In K. D.C. & R. W. (Eds.), *Perception as Bayesian Inference*. Cambridge, U.K.: Cambridge University Press.
- Yuille, A., Coughlan J., Kersten D. (1998) (pdf)