Slide 2:

Imagine you’re viewing this image and you have to estimate what the orientations are. You might want to do this for a number of reasons like determining whether a branch looks like it might fall on you. Our brain is able to quickly and accurately estimate many orientations simultaneously, even when they are overlapping. For example, you can easily separate out these two lines and see that they are -20 and 20

Slide 3:

Lets think for a minute about how the brain does this. In many popular neural coding models, each neuron has a preferred stimulus. The neural responses are variable though: when looking at a stimulus s, a neuron will produce a number of spikes with a mean given by the height of the tuning curve on the graph. Here we are going to assume that the spikes form a Poisson distribution with that mean

Slide 4:

Under those assumptions, here’s an example of how a population of neurons with evenly spaced tuning curves might respond to a line tilted at -20 degrees, that branch from the picture. And similarly, here’s how it might respond to a 20 degree branch. But things get more complicated when there is overlap. Under many models of coding, the response to two stimuli in the receptive field is a weighted sum of the responses to each individual stimulus. We call this stimulus mixing. In order to figure out what the actual stimuli out in the world are, a downstream brain area must “demix” these responses. This seems fairly straightforward here.

Slide 5:

But lets imagine what happens if the stimuli are closer together. Now it’s a little more difficult to read-out what the original stimuli were.

Slide 6:

And what if the contrast in the stimulus is lower. In most models, this decreases the gain of the neural response so there are fewer spikes on an average trial. This makes demixing even harder.

Slide 7:

So demixing can be a hard problem for the brain. And in fact it is much more pervasive than it orientation, there is lots of evidence to suggest that coding of motion in MT and objects in IT, among many other brain areas, have mixed stimuli representations. So what is a poor brain to do if it wants to solve tasks? The best thing to do would be to do Bayesian inference. That is: get the probability distribution over the stimuli given the neural response you are trying to decode and use the mean of that. If we can write down a probability model for the neural responses, we can automatically handle any stimulus pair and any combination of contrasts *optimally*. In addition, we might be interested in not only estimating but also our confidence which is encoded in the width of that distribution. Or we might want to figure out the probability that a branch is going to hit us on the head which is dependent on the probabilities of the angles of the branches. Let’s see what the posterior looks like in some of the examples we’ve already seen.

Slide 8:

Slide 9:

So Bayesian inference is great and as people with brains, it would be great if our brains did it.

Slide 10:

But in complex situations, it’s pretty much impossible. There is lots of people in machine learning and statistics departments trying to figure out how to get reasonable approximations and there has been some work trying to see if the brain uses these.

Slide 11:

But the brain actually already has some very powerful computational abilities. We know (to a gross approximation) that many neurons in cortex have firing rates that are thresholded linear sums of their inputs. And it turns out that groups of these approximate neurons can do state of the art computation in many machine learning problems. In fact, theorems have shown that there exists a set of weights such that a network with two layers with a finite number of these type of neurons can approximate any function arbitrarily well. So maybe we can just use the computational power that the brain already has to approximate Bayesian inference?

Slide 12:

In order to test this, we started with 61 input neurons. These represent the input population with the mixed stimulus representation from earlier. These were generated with this function. On top of that we use a layer of 20 hidden units that compute the rectified linear sums. Finally, we have two output neurons that compute linear sums of the hidden layer. In order to find the right weights for this problem, we gave the network 270,000 examples of input responses. The network would estimate s\_1 and s\_2 for that input and then we gave it an error signal which was the Mean Squared Error of between it’s estimate and the true s\_1 and s\_2 that generated the input. We used SGD with backprop to minimize that cost function.

Slide 13:

To start investigating how well this works, we picked a set of gains and gave the network training examples with only that set of gains. We then tested the network with a bunch of trials generated with that set of gains. We also computed what the posterior mean was on all of those trials to use as our target. In these cases, both the network and the posterior know what the gains in the trial are going to be. A lot of the interesting posteriors occur when the two stimuli are very close to each other so I’m going to show some plots where we set s\_1 to -30 and vary s\_2 from -30 to 0. In order to get a sense of how well the network is doing, we split up the errors into a bias term: how far the mean estimate of s\_2 is from the original s\_2 and a standard deviation: how variable that estimate is. Since s\_2 is varying, we are going to be plotting the s\_2 estimate

Slide 14 (FIX PLOTS):

We start with both gains = 1. Here is what the posterior mean bias and standard deviation is on these trials. Since you can get different networks every time you use a different training set ordering, we trained 200 networks and here we are plotting plus/minus one SD of the bias and SD of those networks estimates. And here is what it looks like as you increase gain. In addition, we can see what it looks like when the two stimuli have different gains.

Slide 15:

However, in the real world, it is not the case that there is only one contrast, you experience a wide range of contrasts. So lets see what happens when we train the network with all combinations of 1, 2, 4 and then test it on that. For plotting, we’ll split it up into different gain combinations.

Slide 16:

Here are the same plots as above for this set of networks.

Slide 17 (FIX THESE):

Ok, you might be saying at this point is this really so interesting? We already have that function-approximating theorem; we know a neural network can approximate any function. This isn’t exactly true - while the theorem says that there are weights, it doesn’t say that they will be easy to find or that 20 units is enough for a good approximation. But this is a pretty easy problem so maybe it’s not that surprising that it works pretty well. But what about for examples the network has never seen before. What if we only train the network on 1, 1 and 4, 4 and see what happens?

Slide 18:

Here it is broken out by gain like above. It seems to do a pretty reasonable job.

Slide 19 (ADD POSTERIOR PLOT):

But posteriors are more than just a point estimate. They let us read out other quantities related to whatever task we might be doing. For example we might want to know how confident we are in our estimate, which we might think is related to the width of our posterior. Now that the neural network can get reasonable estimates, maybe it automatically learned something about the whole posterior, even if we didn’t actually train it to do that. Let’s see if we can decode the posterior precision from the hidden units.

Slide 20:

We’ll start by looking at the network trained with all contrast combinations. This allowed it to see a wide variety of different posterior widths so it would be the most likely to work. Many constructed models of neural codes with specific computations/weights allow you to recover the precision by just summing the activations of the hidden units. So lets see if that works here. Nope.

Slide 21:

But maybe if we do a weighted sum it’ll work. Ah correlation of .83!

Slide 22:

But how good is this really? If we train a neural network to go from the inputs directly to precisions, wouldn’t that do better? Turns out not significantly better.

Slide 23:

And it turns out that the 1,1 4, 4 network doesn’t do much worse, even though it didn’t see as wide of a range of trials.

Slide 24:

And the just 4, 4 network actually is still pretty good.

Slide 25