MICHALE FEE: All right, let's go ahead and get started. So we're starting a new topic today. This is actually one of my favorite lectures, one of my favorite subjects in computational neuroscience. All right, so brief recap of what we've been doing. So we've been working on circuit models of neural networks. And we've been working on what we call a rate model, in which we replaced all the spikes of a neuron with, essentially, a single number that characterizes the rate at which a neuron fires.

We introduced a simple network in which we have an input neuron and an output neuron with a synaptic connection of weight w between them. And that synaptic connection leads to a synaptic input that's proportional to w times the firing rate of the input neuron. And then we talked about how we can characterize the output, the firing rate of the output neuron, as some nonlinear function of the total input to this output neuron.

We've talked about different F-I curves. We've talked about having what's called a binary threshold unit, which has zero firing below some threshold. And then actually, there are different versions of the binary threshold unit. Sometimes the firing rate is zero for inputs below the threshold. And in other models, we use a minus 1. And then a constant firing rate of one above that threshold.

And we also talked about linear neurons, where we can write down the firing rate of the output neuron just as a weighted sum of the inputs. And remember that these neurons are kind of special in that they can have negative firing rates, which is not really biophysically plausible, but mathematically, it's very convenient to have neurons like this.

So we took this simple model and we expanded it to the case where we have many input neurons and many output neurons. So now we have a vector of input firing rates, u, and a vector of output firing rates, u. And for the case of linear neurons, we talked about how you can write down the vector of firing rates of the output neuron simply as a matrix product of a weight matrix times the vector of input firing rates. And we talked about how this can produce transformations of this vector of input firing rates.

So in this high-dimensional space of inputs, we can imagine stretching that input vector along different directions to amplify certain directions that may be more important than others. We talked about how you can do that, stretch in arbitrary directions, not just along the axes. And we talked about how that vector of-- that, sorry, matrix of weights can produce a rotation.

So we can have some set of inputs where, let's say, we have clusters of different input values corresponding to different things. And you can rotate that to put certain features in particular output neurons. So now you can discriminate one class of objects from another class of objects by looking at just one dimension and not the whole high-dimensional space.

So today, we're going to look at a new kind of network called a recurrent neural network, where not only do we have inputs to our output neurons from an input layer, but we also have connections between the neurons in the output layer. So these neurons in a recurrent network talk to each other. And that imbues some really cool properties onto these networks.

So we're going to develop the math and describe how these things work to develop an intuition for how recurrent networks respond to their inputs. We're going to get into some of the computations that recurrent networks can do. They can act as amplifiers in particular directions. They can act as integrators, so they can accumulate information over time. They can generate sequences. They can act as short-term memories of either continuous variables or discrete variables. It's a very powerful kind of circuit architecture.

And on top of that, in order to describe these mathematically, we're going to use all of the linear algebra tools that we've been developing so far. So, hopefully, a bunch of things will kind of connect together. OK, so mathematical description of recurrent networks. We're going to talk about dynamics in these recurrent networks, and we're going to start with the very simplest kind of recurrent network called an autapse network. Then we're going to extend that to the general case of recurrent connectivity.

And then we're going to talk about how recurrent networks store memories. So we'll start talking about a specific circuit models for storing short-term memories. And I'll

touch on recurrent networks for decision-making. And this will kind of lead into the last few lectures of the class, where we get into how sort of specific cases of looking at how networks can store memories.

OK, mathematical description. All right, so the first thing that we need to do is-- the really cool thing about recurrent networks is that their activity can evolve over time. So we need to talk about dynamics, all right? The feed-forward networks that we've been talking about, we just put in an input. It gets weighted by synaptic strength, and we get a firing rate in the output, just sort of instantaneously. We've been thinking of you put an input, and you get an output.

In general, neural networks don't do that. You put an input, and things change over time until you settle at some output, maybe, or it starts doing something interesting, all right? So the time course of the activity becomes very important, all right? So neurons don't respond instantaneously to inputs. There are synaptic delays. There are integration of membrane potential. Things change over time.

And a specific example of this that we saw in the past is that if you have an input spike, you can produce a postsynaptic current that jumps up abruptly as the synaptic conductance turns on. And then the synaptic conductance decays away as the neurotransmitter unbinds from the neurotransmitter receptor, and you get a synaptic current that decays away over time, OK? So that's a simple kind of time dependence that you would get.

And that could lead to time dependence in the firing rate of the output neuron. OK, dendritic propagation, membrane time constant, other examples of how things can take time in a neural network. All right, so we're going to model the firing rate of our output neuron in the following way. If we have an input firing rate that's zero and then steps up to some constant and then steps down, we're going to model the output, the firing rate of the output neuron, using exactly the same kind of first order linear differential equation that we've been using all along for the membrane potential, for the Hodgkin-Huxley gating variables. The same kind of differential equation that you've seen over and over again.

So that's the differential equation we're going to use. We're going to say that the time derivative of the firing rate of the output neuron times the time constant is just

equal to minus the firing rate of the output non plus v infinity. And so you know that the solution to this equation is that the firing rate of the output neuron will just relax exponentially to some new v infinity.

And the v infinity that we're going to use is just this non-linear function times the weighted input to our neuron. So we're going to take the formalism that we developed for our feed-forward networks to say, what is the firing rate of the output neuron as a function of the inputs? And we're going to use that firing rate that we've been using before as the v infinity for our network with dynamics. Any questions about that?

All right, so that becomes our differential equation now for this recurrent network, all right? So it's just a first order linear differential equation, where the v infinity, the steady state firing rate of the output neuron, is just this nonlinear function times the weighted sum of all the inputs. All right, and actually, for most of what we do today, we're going to just take the case of a linear neuron.

All right. So this I've already said. This I've already said. And actually, what I'm doing here is just extending this. So this was the case for a single output neuron and a single input neuron. What we're doing now is we're just extending this to the case where we have a vector of input neurons with a firing rate represented by a firing rate vector u, and a vector of output neurons with a fine rate vector v. And we're just going to use this same differential equation, but we're going to write it in vector notation.

So each one of these output neurons has an equation like this, and we're going to combine them all together into a single vector. Does that make sense? All right, so there is our vector notation of the activity in this recurrent network. Sorry, I forgot to put the recurrent connections in there.

So the time dependence is really simple in this feed-forward network, right? So in a feed-forward network, the dynamics just look like this. But in a recurrent network, this thing can get really interesting and start doing interesting stuff.

All right, so let's add recurrent connections now and add these recurrent connections to our equation. So in addition to this weight matrix w that describes the connections from the input layer to the output layer, we're going to have

another weight matrix that describes the connections between the neurons in the output layer. And this weight matrix, of course, has to be able to describe a connection from any one of these neurons to any other of these neurons.

And so this weight matrix is going to be a function of the postsynaptic neuron, the weight-- the synaptic strength is going to be a function of the postsynaptic neuron and the presynaptic-- the identity of the postsynaptic neuron and the identity of the presynaptic neuron. Does that make sense?

OK, so there are two kinds of input-- a feed-forward input from the input layer and a recurrent input due to connections within the output layer. Any questions about that? OK, so there is the equation now that describes the time rate of change of the firing rates in the output layer. It's just this first order linear differential equation. And the infinity is just this non-linear function of the inputs, of the net input to this neuron, to each neuron.

And the net input to this set of neurons is a contribution from the feed-forward inputs, given by this weight matrix w, and this contribution from the recurrent inputs, given by this weight matrix, m. So that is the crux of it, all right? So I want to make sure that we understand where we are. Does anybody have any questions about that? No? All right, then I'll push ahead.

All right, so what is this? So we've seen this before. This product of this weight matrix times this vector of input firing rates just looks like this. You can see that the input to this neuron, this first output neuron, is just the dot product of these weights onto the first neuron and the dot product of that vector of weights, that row of the weight matrix, with the vector of input firing rates.

And the feed-forward contribution to this neuron is just the dot product of that row weight of this input weight matrix with the vector of input firing rates, and so on. If we look at the recurrent input to these neurons, the recurrent input to this first neuron is just going to be the dot product of this row of the recurrent weight matrix and the vector of firing rates in the output layer.

The recurrent inputs to the second neuron is going to be the dot product of this row of the weight matrix and the vector of firing rates. Yes?

AUDIENCE: So I guess I'm a little confused, because I thought it was from A. Oh, to A. OK.

MICHALE FEE: Yeah, it's always post, pre. Post, pre in a weight matrix. That's because we're usually writing down these vectors the way that I'm defining this notation. This vector is a column matrix, a column vector.

All right, so we're going to make one simplification to this. When we work with the recurrent networks, we're usually going to simplify this input. And rather than write down this complex feed-forward component, writing this out as this matrix product, we're just going to simplify the math. And rather than carry around this w times u, we're just going to replace that with a vector of inputs onto each one of those neurons, OK?

So we're just going to pretend that the input to this neuron is just coming from one input, OK? And the input to this neuron is coming from another single input. And so we're just going to replace that feed-forward input onto this network with this vector h. So that's the equation that we're going to use moving forward, all right? Just simplifies things a little bit so we're not carrying around this w u.

So now, that's our equation that we're going to use to describe this recurrent network. This is a system of coupled equations. What does that mean? You can see that the time derivative of the firing rate of this first neuron is given by a contribution from the input layer and a contribution from other neurons in the output layer. So the time rate of change of this neuron depends on the activity in all the other neurons in the network. And the time rate of change in this neuron depends on the activity of all the other neurons in the network.

So that's a set of coupled equations. And that, in general, can be-- you know, it's not obvious, when you look at it, what the solution is, all right? So we're going to develop the tools to solve this equation and get some intuition about how networks like this behave in response to their inputs.

So the first thing we're going to do is to simplify this network to the case of linear neurons. So we don't have-- so the neurons just fire. Their firing rate is just linear with their input. And so that's the equation for the linear case. All we've done is we've just gotten rid of this non-linear function f.

All right, so now let's take a very simple case of a recurrent network and use this equation to see how it behaves, all right? So the simplest case of a recurrent network is the case where the recurrent connections within this layer are given by-- the weight matrix is given by a diagonal matrix. Now, what does that correspond to? What that corresponds to is this neuron making a connection onto itself with a synapse of weight lambda one, right there.

And that kind of recurrent connection of a neuron onto itself is called an autapse, like an auto synapse. And we're going to put one of those autapses on each one of these neurons in our output layer, in our recurrent layer. So now we can write down the equation for this network, all right? And what we're going to do is simply replace-- sorry, let me just bring up that equation again. Sorry, there's the equation.

And we're simply going to replace this weight matrix m, this recurrent weight matrix, with that diagonal matrix that I just showed you. So there it is. So that time rate of change of this vector of output neurons is just minus v plus this diagonal matrix times [INAUDIBLE] plus the inputs.

So now you can see that if we write out the equation separately for each one of these output neurons-- so here it is in vector notation. We can just write that out for each one of our output neurons. So there's a separate equation like this for each one of these neurons. But you can see that these are all uncoupled. So we can understand how this network responds just by studying this equation for one of those neurons.

OK, so let's do that. We have an independent equation. The firing rate change-- the time derivative of the firing rate of neuron one depends only on the firing rate of neuron one. It doesn't depend on any other neurons. As you can see, it's not connected to any of the other neurons.

OK, so let's write this equation. And let's see what that equation looks like. So we're going to rewrite this a little bit. We're just going to factor out the va all right here. This parameter, 1 minus lambda a, controls what kind of solutions this equation has. And there are three different cases that we need to consider. We need to consider the case where 1 minus lambda is greater than zero, equal to zero, or less than zero.

Those three different values of that parameter 1 minus lambda give three different kinds of solutions to this equation. We're going to start with the case where lambda is less than one. And if lambda is less than 1, then this term right here is greater than zero. If we do that, then we can rewrite this equation as follows.

We're going to divide both sides of this equation by 1 minus lambda, and that's what we have here. And you can see that this equation starts looking very familiar, very simple. We have a first order linear differential equation, where we have a time constant here, tau over 1 minus lambda, and a v infinity here, which is the input, the effective input onto that neuron, divided by 1 minus lambda. So that's tau dv dt equals minus v plus v infinity.

But now you can see that the time constant and the v infinity depend on lambda, depend on the strength of that connection, all right? And the solution to that we've seen before, to this equation. It's just exponential relaxation toward v infinity.

OK, so here's our v infinity. There's our tau. True for the case of lambda between-- let's just look at these solutions for the case of lambda between zero and one. So I'm going to plot v as a function of time when we have an input that goes from zero and then steps up and then is held constant.

All right, so let's look at the case of lambda equals zero. So this lambda zero means there's no autapse. It's just not connected. So you can see that, in this case, the solution is very simple. It's just exponential relaxation toward infinity. v infinity is just given by h, the input, and tau is just the original tau, 1 minus 0, right? So it's just exponential relaxation to h. That make sense?

And it relaxes with a time constant tau, tau m. We're going to now turn up the synapse a little bit so that it has a little bit of strength. You see that what happens when lambda is 0.5, that v infinity gets bigger. v infinity goes to 2h. Why? Because it's h divided by 1 minus 0.5. So it's h over 0.5, so 2h. And what happens to the time constant? Well, it becomes two tau.

All right, and if we make lambda equal to 0.3-- sorry, 0.66. We turn it up a little bit. You can see that the response of this neuron gets even bigger. So you can see that what's happening is that when we start letting this neuron feed back to itself, positive feedback, the response of the neuron to a fixed input-- the input is the

same for all of those. The response of the neuron gets bigger. And so having positive feedback of that neuron onto itself through an autapse just amplifies the response of this neuron to its input.

Now, let's consider the case where-- so positive feedback amplifies the response. And what also does it do? It slows the response down. The time constants are getting longer, which means the response is slower. All right, let's look at what happens when the lambdas are less than zero. What does lambda less than zero correspond to here?

AUDIENCE: [INAUDIBLE]

MICHALE FEE: Yeah, which is, in neurons, what does that correspond to?

AUDIENCE: [INAUDIBLE]

MICHALE FEE: Inhibition. So this neuron, when you put an input in, it tries to activate the neuron. But that neuron inhibits itself. So what do you think's going to happen? So positive feedback made the response bigger. Here, the neuron is kind of inhibiting itself. So what's going to happen? You put in that same h that we had before, what's going to happen when we have inhibition?

AUDIENCE: Response is [INAUDIBLE].

MICHALE FEE: What's that?

AUDIENCE: The response is going to be smaller.

MICHALE FEE: The response will just be smaller, that's right. So let's look at that. So here's firing rate of this neuron is a function of time for a step input. You can see for a lambda equals zero, we're going to respond with an amount h. But if we put in-- in a time constant tau. If we put in a lambda of negative one-- that means you put this input in-- that neuron starts inhibiting itself, and you can see the response is smaller.

But another thing that's real interesting is that you can see that the response of the neuron is actually faster. So if the feedback-- if the lambda is minus one, you can see that v infinity is h over 1 minus negative 1. So it's h over 2. All right, and so on. The more we turn up that inhibition, the more suppressed the neuron is, the weaker

the response that neuron is to its input, but the faster it is. So negative feedback suppresses the response of the neuron and speeds up the response.

OK, now, there's one other really important thing about recurrent networks in this regime, where this lambda is less than one. And that is that the activity always relaxes back to zero when you turn the input off. OK, so you put a step input in, the neuron responds, relaxing exponentially to sum of v infinity. But when you turn the input off, the network relaxes back to zero, OK?

So now let's go to the more general case of recurrent connections. Oh, and first, I just want to show you how we actually show graphically how a neuron responds-- sorry, how one of these networks respond. And a typical way that we do that is we plot the firing rate of one neuron versus the firing rate of another neuron. That's called a state-space trajectory. And we plot that response as a function of time after we put in an input.

So we can put an input in described as some vector. So we put in some h1 and h2, and we then plot the response of the neuron-- the response of the network in this output state space. So let me show you an example of what that looks like. So here is the output of this little network for different kinds of inputs. So Daniel made this nice little movie for us.

Here, you can see that if you put an input into neuron one, neuron one responds. If you put a negative input into neuron one, the neuron goes negative. If you put an input into neuron two, the neuron responds. And if you put a negative input into neuron two, it responds. Now, why did it respond bigger in this direction than in this direction?

AUDIENCE: That's [INAUDIBLE].

MICHALE FEE: Good. Because neuron one had--

AUDIENCE: Positive?

MICHALE FEE: Positive feedback. And neuron two had negative feedback. So neuron one, this neuron one, amplified its input and gave a big response. Neuron two suppressed the response to its input, and so it had a weak response.

Let's look at another interesting case. Let's put an input into these neurons-- not one at a time, but simultaneously. So now we're going to put an input into both neurons one and two simultaneously. It's like Spirograph. Did you guys play with Spirograph? It's kind of weird, right? It's like making little butterflies for spring.

So why does the output-- why does the response of this neuron to an input, positive input to both h1 and h2, look like this? Let's just break this down into one of these little branches. We start at zero. We put an input into h1 and h2, and the response goes quickly like this and then relaxes up to here. So why is that? Lena?

AUDIENCE: [INAUDIBLE] so there was [INAUDIBLE] and then because it's negative, it's shorter.

MICHALE FEE: Yup. The response in the v2 direction is weak but fast.

AUDIENCE: Yeah.

MICHALE FEE: So it goes up quickly. And then the response in the v1 direction is?

AUDIENCE: Slow, but [INAUDIBLE].

MICHALE FEE: Good. That's it. It's slow, but [AUDIO OUT]. It's amplified in this direction, suppressed in this direction. But the response is fast this way and slow this way. So it traces this out. Now, when you turn the input off, again, it relaxes. v2 relaxes quickly back to zero, and v1 relaxes slowly back to zero. So it kind of traces out this kind of hysteretic loop. It's not really hysteresis.

Then it's exactly mirror image when you put in a negative input. And when you put in h1 positive and v1 negative, it just looks like a mirror image. All right, so any questions about that? Yes, Lena?

AUDIENCE: If there was nothing, like no kind of amplified or [INAUDIBLE], would it just be like a [INAUDIBLE]?

MICHALE FEE: Yeah, so if you took out the recurrent connections, what would what would it look like?

AUDIENCE: An x?

MICHALE FEE: Yeah, the output-- so let's say that you just literally set those to zero. Then the

response will be the identity matrix, right? You get the output as a function of input. Let's just go back to the equation. Can always, always get the answer by looking at the equation. Too many animations.

No, it's a very good question. Here we go. There it is right there. So you're asking about-- let's just ask about the steady state response. So we can set dv dt equal to zero. And you're asking, what is v? And you're saying, let's set lambda to zero, right? We're going to set all these diagonal elements to zero. And so now v equals h.

OK, great question. Now, let's go to the case of fully recurrent networks. We've been working with this simplified case of just having neurons have autapses. And the reason we've been doing that is because the answer you get for the autapse kind of captures almost all the intuition that you need to have. What we're going to do is we're going to take a fully recurrent neural network, and we're going to do a mathematical trick that just turns it into an autapse network.

And the answer for the fully recurrent network is just going to be just as simple as what you saw here. All right, so let's do that. Let's take this fully recurrent network. Our weight matrix m now, instead of just having diagonal elements, also has off-diagonal elements.

And I'll say that one of the things that we're going to do today is just consider the simplest case of this fully recurrent network, where the connections are symmetric, where a connection from v1 to v2 is equal to the connection from v2 to v1, all right? We're going to do that because that's the next thing to do to build our intuition, and it's also mathematically simpler than the fully general case, OK?

So we saw how the behavior of this network is very simple if m is diagonal. So what we're going to do is we're going to take this arbitrary matrix m, and we're going to just make it diagonal. So let's do that. So we're going to rewrite our weight matrix m as-- so we're going to rewrite m in this form, where this phi-- sorry, where this lambda is a diagonal matrix.

So we're going to take this network with recurrent connections between different neurons in the network, and we're going to transform it into sort of an equivalent network that just has autapses. So how do we write m in this form, with a rotation matrix times a diagonal matrix times a rotation matrix? We just solve this

eigenvalue equation, OK? Does that make sense?

We're just going to do exactly the same thing we did in PCA, where we find the covariance matrix. And we rewrote the covariance matrix like this. Now we're going to take a weight matrix of this recurrent network, and we're going to rewrite it in exactly the same way. So that process is called diagonalizing the weight matrix.

So the elements of lambda here are the eigenvalues of m. And the columns of the phi are the eigenvectors of m. And we're going to use these quantities, these elements, to build a new network that has the same properties as our recurrent network. So let me just show you how we do that.

So remember that what this eigenvalue-- this is an eigenvalue equation written in matrix notation. What this means is this is set of eigenvalues equations that have-- it's a set of n eigenvalue equations like this, where there's one of these for each neuron in the network. OK, so let me just go through that.

OK, so here's the eigenvalue equation. If M is a symmetric matrix, then the eigenvalues are real and phi is a rotation matrix. And the eigenvectors give us an orthogonal basis, all right? So everybody remember this from a few lectures ago?

If M is symmetric-- and this is why we're going to, at this point on, consider just the case where M is symmetric, then the eigenvectors, the columns of that matrix phi, give us an orthogonal set of vectors and their unit vectors. So it satisfies this orthonormal condition. And phi transpose phi is an identity matrix, which means phi is a rotation matrix.

OK, so now what we're going to do is rewrite. The first thing we're going to do to use this trick to rewrite our matrix, our network, is to rewrite the vector of firing rates v in this new basis. What are we going to do? Well take the vector and all we're going to do is to rewrite that vector in this new basis set. We're just going to do a change of basis of our firing rate vector into a new basis set that's given by the columns of phi.

Another way of saying it is that we're going to rotate this firing rate vector v using the phi rotation matrix. So we're going to project v onto each one of those new basis vectors. So there's v in the standard basis. There's our new basis, f1 and f2. We're

going to project v onto f1 and f2 and write down that scalar projection, c1 and c2. So we're going to write down the scalar projection of v onto each one of those basis vectors.

So we can write that c sub alpha-- that's the alpha-th component-- is just v dot the alpha-th basis vector. So now we can express v as a linear combination in this new basis. So it's c1 times f1 plus c2 times f2 plus c3-- that's supposed to be a three-- times f3 and so on.

And of course, remember, we're doing all of this because we want to understand the dynamics. So these things are time dependent. So v is v changes in time. We're not going to be changing our basis vectors in time. So if we want to write down a time dependent v, it's really these coefficients that are changing in time, right? Does that make sense?

So we can now write our vector v, our firing rate vector, as a sum of contributions in all these different directions corresponding to the new basis. And each one of those coefficients, c is just the time dependent v projected onto one of those basis vectors. And questions? No? OK.

And remember, we can write that in matrix notation using this formalism that we developed in the lecture on basis sets. So v is just phi c, and c is just phi transpose v. So we're just taking this vector v, and we're rotating it into a new basis set, and we can rotate it back.

All right, so now what we're going to do is we're going to take this v expressed in this new basis set and were going to rewrite our equation in that new basis set. Watch this. This is so cool. All right, you ready? We're going to take this, and we're to plug it into here. So dv dt is phi dc dt. V is just phi c. v is phi c, and h doesn't change.

So now what is that? Do you remember?

AUDIENCE: Phi [INAUDIBLE].

MICHALE FEE: Right. We got phi as the solution to the eigenvalue equation. What was the eigenvalue equation? The eigenvalue equation was m phi equals phi lambda. So the phi here, this rotation matrix, is the solution to this equation, all right? So we're given m, and we're saying we're going to find a phi and a lambda such that we can

write m phi is equal to phi lambda.

So when we take that matrix m and we run eig on it in Matlab, Matlab sends us back a phi and a lambda such that this equation is true. So literally, we can take the weight matrix m stick it into Matlab, and get a phi and a lambda such that m phi is equal to phi lambda. So m phi is equal to what? Phi lambda. That becomes this.

Now, all of a sudden, this thing is just going to simplify. So how would we simplify this equation? We can get rid of all of these things, all of these phi's, by doing what? How do you get rid of phi's?

AUDIENCE: Multiply [INAUDIBLE] phi transpose.

MICHALE FEE: You multiply by phi transpose, exactly. So we're going to multiply each term in this equation by phi transpose. So what do you have? Phi transpose phi, phi transpose phi, phi transpose phi. What is phi transpose phi equal to? The identity matrix. Because it's a rotation matrix, phi transpose is just the inverse of phi. So phi inverse phi is just equal to the identity matrix. And all those things disappear. And you're left with this equation-- tau dc dt equals minus c plus lambda c plus h, hf.

And what is hf? hf is just h rotated into the new basis set. So this is the equation for a recurrent network with just autapses, which we just understood. We just wrote down what the solution is, right? And we plotted it for different values of lambda.

So now let's just look at what some of these look like. So we've rewritten our weight matrix in a new basis set. We've rebuilt our network and a new basis set, in a rotated basis set where everything simplifies. So we've taken this complicated network with recurrent connections and we've rewritten it in a new network, where each of these neurons in our new network corresponds to what's called a mode of the fully recurrent network.

So the activities c alpha c1 and c2 of the network modes represent kind of an activity in a linear combination of these neurons. So we're going to go through what that means now. So the first thing I want to do is just calculate what the steady state response is in this neuron. And I'll just do it mathematically, and then I'll show you what it looks like graphically.

So there's our original network equation. We've rewritten it a set of differential equations for the modes of this network. I'm just rewriting this by putting an I here, minus I times c. That's the only change I made here. I just rewrote it like this.

Let's find a steady state. So we're going to set dc dt equal to zero. We're going to ask, what is c in steady state? So we're going to call that c infinity, all right? I minus lambda times c infinity equals phi transpose h. OK, don't panic. It's all going to be very simple in a second. c infinity is just I minus lambda inverse phi transpose h.

But I is diagonal. Lambda is diagonal. So I minus lambda inverse is just the-- it's a diagonal matrix with these elements with one over all those diagonal elements. Now let's calculate v infinity. v infinity is just phi times v infinity. So here, we're multiplying on the left by phi. That's just v infinity. So v infinity is just this.

So what is this? This just says v infinity is some matrix-- it's a rotated stretch matrix-- times the input. So v infinity is just this matrix times h. And now let's look at what that is. v infinity is a matrix times h. We're going to call that g. v infinity is a gain matrix. We're going to think of that as a gain times the input. So it's just a matrix operation on the input.

This matrix has exactly the same eigenvectors as m. And the eigenvalues are just 1 over 1 minus lambda. Hang in there. So what this means is that if an input is parallel to one of the eigenvectors of the weight matrix, that means the output is parallel to the input.

So if the input is in the direction of one of the eigenvectors, v infinity is g times f. But g times f-- f is an eigenvector v. And what that means is that v infinity is parallel to f with a scaling factor 1 over 1 minus lambda. All right? So hang in there. I'm going to show you what this looks like.

So in steady state, the output will be parallel to the input if the input is in the direction of one of the eigenvectors of the network. So if the input is in the direction of one of the eigenvectors of the network, that means you're activating only one mode of the network. And only that one mode responds, and none of the other modes respond.

The response of the network will be in the direction of that input, and it will be

amplified or suppressed by this gain factor. And the time constant will also be increased or decreased by that factor. So now let's look at-- so I just kind of whizzed through a bunch of math. Let's look at what this looks like graphically for a few simple cases. And then I think it will become much more clear.

Let's just look at a simple network, where we have two neurons with an excitatory connection from neuron one to neuron two, an excitatory connection from neuron two to neuron one. And we're going to make that weight 0.8. OK, so what is the weight matrix M look like? Just tell me what the entries are for M.

AUDIENCE: Does it not have the autapse?

MICHALE FEE: No, so there's no connection of any of these neurons onto themselves.

AUDIENCE: So you have, like, zeros on the diagonal.

MICHALE FEE: Zeros on the diagonal. Good.

AUDIENCE: All the diagonals.

MICHALE FEE: Good. Like that? Good. Connection from neuron one to itself is zero. The connection from post, pre is row, column. So onto neuron one from neuron two is 0.8. Onto neuron two from neuron one is 0.8. And neuron two onto neuron two is zero.

So now we are just going to diagonalize this weight matrix. We're going to find the eigenvectors and eigenvalues. The eigenvectors are the columns of phi. And the eigenvalues are the diagonal elements of lambda. Let's take a look at what those eigenvectors are. So this vector here is f1. This vector here is another eigenvector, f2.

And how did I get this? How did I get this from this? How would you do that? If I gave you this matrix, how would you find phi?

AUDIENCE: Eig M.

MICHALE FEE: Good, eig of M. Now, remember in the last lecture when we were talking about some simple cases of matrices that are really easy to find the eigenvectors of? If you have a symmetric matrix, where the diagonal elements are equal to each other, the eigenvectors are always 45 degrees here and 45 degrees there. And the

eigenvalues are just the diagonal elements plus or minus the off-diagonal elements.

So the eigenvalues here are 0.8 and minus 0.8. All right, so those are the two eigenvectors of this matrix, of this network. Those are the modes of the network. Notice that one of the modes corresponds to neuron one and neuron two firing together. The other mode corresponds to neuron one and neuron two firing with opposite sign-- minus one, one.

So the lambda-- the diagonal elements of the lambda matrix are the eigenvalues. They're 0.8 and minus 0.8, a plus or minus b. Now, this gain factor, what this says is that if I have an input in the direction of f1, the response is going to be amplified by a gain. And remember, we just derived, on the previous slide, that that gain factor is just 1 over 1 minus the eigenvalue for that eigenvector.

In this case, the eigenvalue for mode one is 0.8. So 1 over 1 minus 0.8 is 5. So the gain in this direction is 5. The gain for an input in this direction is 1 over 1 minus negative 0.8, which is 1 over 1.8. Does that makes sense? OK, let's keep going, because I think it will make even more sense once we see how the network responds to its inputs.

So zero input. Now we're going to put an input in the direction of this mode one. And you can see the mode responds a lot. Put a negative input in, it responds a lot. If we put a mode input in this direction or this direction, the response is suppressed by an amount of about 0.5. Because here, the gain is small. Here, the gain is big.

So you see what's happening? This network looks just like an autapse network, but where we've taken this input and output space and just rotated it into a new coordinate system, into this new basis. Yes?

AUDIENCE: Why did it kind of loop around on the one side [INAUDIBLE]?

MICHALE FEE: OK, it's because these things are relaxing exponentially back to zero. And we got a little bit impatient and started the next input before it had quite gone away. OK, good question. It's just that if you really wait for a long time for it to settle, then the movie just takes a long time. But maybe it would be better to do that.

So input this way and this way lead to a large response, because those inputs activate mode one, which has a big gain. Inputs in this direction and this direction

have a small response, because they activate mode two, which has small gain. But notice that when you activate mode one-- when you put an input in this direction, it only activates mode one. And it doesn't activate mode two at all.

If you put an input in this direction, then it only activates mode two, and it doesn't activate mode one at all. So it's just like the autapse network, but rotated. So now let's do the case where we have an input that activates both modes. So let's say we put an input in this direction. What does that direction correspond to h up. What is that input mean here in terms of h1 and h2?

Let's say we just put an input-- remember, this is a plot on axes h1 versus h2. So this input vector h corresponds to just putting an input on h2, into this neuron. So you can see that when we put an input in this direction, we're activating-- that input has a projection onto mode one and mode two. So we're activating both modes. You can see that the input h has a projection onto f1 and projection onto f2.

So what you do is-- well, here, I'm just showing you what the steady state response is mathematically. Let me just show you what that looks like. What this says is that if we put an h in this direction, it's going to activate a little bit of mode one with a big gain and a little bit of mode two with a very small gain. And so the steady state response will be the sum of those two. It'll be up here.

So the steady state response to this input in this direction is going to be over here. Why? Because that input activates mode one and mode two both. But the response of mode one is big, and the response of mode two is really small. And so the steady state response is going to be way over here because of the big response, the amplified response of mode two, which is in this direction, OK?

So when we put an input straight up, the response of the network's going to be all the way over here. How is it going to get there? Let's take a look. We're going to put an input-- sorry, that was first in this direction. Now let's see what happens when we put an input in this direction. You can see the response is really big along the mode one direction, in this direction, and it's really small in this direction.

So input up in the upward direction onto just this neuron produces a large response in mode, which is this way, and a very small response in mode two, which is this way.

The response in mode two is very fast, because the lambda, the 1 over 1 minus lambda, is small, which makes the time constant faster and the response smaller.

So, again, it's just like the response of the autapse network, but rotated into a new coordinate system. All right, any questions about that? So you can see we basically understood everything we needed to know about recurrent networks just by understanding simple networks with just autapses. And all these more complicated networks are just nothing but rotated versions of the response of a network with just autapses.

Any questions about that? OK, let's do another network now where we have inhibitory connections. That's called mutual inhibition. And let's make that inhibition minus 0.8. The weight matrix is just zeros on the diagonals, because there's no autapse here. And minus 0.8 on the off-diagonals. What are the eigenvectors for this matrix, for this network?

AUDIENCE: The same.

MICHALE FEE: Yeah, because the diagonal elements are equal to each other, and the off-diagonal elements are equal to each other. It's a symmetric network with equal diagonal elements. The eigenvectors are always at 45 degrees. And what are the eigenvalues?

AUDIENCE: [INAUDIBLE]

MICHALE FEE: Well, the two numbers are going to be the same. It's zero plus and minus 0.8, plus and minus negative 0.8, which is just 0.8 and minus 0.8, right? Good. So the eigenvalues are just 0.8 and minus 0.8. But the eigenvalues correspond to different eigenvectors. So now the eigenvalue mode in the 1, 1 direction is now minus 0.8, which means it's suppressing the response in this direction.

And the eigenvalue for the eigenvector in the minus 1, 1 direction is now close to 1, which means that mode has a lot of recurrent feedback. And so its response in this direction is going to be big. It's going to be amplified. So unlike the case where we had positive recurrent synapses, where we had amplification in this direction, now we're going to have amplification in this direction. Does that make sense?

Think of it this way-- if we go back to this network here, you can see that when

these two neurons-- when this neuron is active, it tends to activate this neuron. And when this neuron is activate, it tends to activate that neuron. So this network, if you were to activate one of these neurons, it tends to drive the other neuron also. And so the activity of those two neurons likes to go together. When one is big, the other one wants to be big.

And that's why there's a lot of gain in this direction. Does that make sense? With these recurrent excitatory connections, it's hard to make this neuron fire and make that neuron not fire. And that's why the response is suppressed in this direction, OK? With this network, when this neuron is active, it's trying to suppress that neuron. When that neuron has positive firing rate, it's trying to make that neuron have a negative firing rate.

When that neuron is negative, it tries to make that one go positive. And so this network likes to have one firing positive and the other neuron going negative. And so that's what happens. What you find is that if you put an input into the first neuron, it tends to suppress the activity in the second neuron, in v2. If you put neuron into neuron two, it tends to suppress the activity, or make v1 go negative.

So it's, again, exactly like the autapse network, but just, in this case, rotated minus 45 degrees instead of plus 45 degrees, OK? Any questions about that? All right. So now let's talk about how-- yes, Linda?

AUDIENCE: So we just did, those were all symmetric matrices, right?

MICHALE FEE: Yes.

AUDIENCE: So [INAUDIBLE] can we not do this strategy if it's not symmetric?

MICHALE FEE: You can do it for non-symmetric matrices, but non-symmetric matrices start doing all kinds of other cool stuff that is a topic for another day. So symmetric matrices are special in that they have very simple dynamics. They just relax to a steady state solution.

Weight matrices that are not symmetric, or even anti-symmetric, tend to do really cool things like oscillating. And we'll get to that in another lecture, all right? OK, so now let's talk about using recurrent networks to store memories.

So, remember, all of the cases we've just described, all of the networks we've just described, had the properties that the lambdas were less than one. So what we've been looking at are networks for which lambda is less than one and they're symmetric weight matrices. So that was kind of a special case, but it's a good case for building intuition about what goes on.

But now we're going to start branching out into more interesting behavior. So let's take a look at what happens to our equation. This is now our equation different modes of a network. What happens to this equation when lambda is actually equal to one?

So when lambda is equal to one, this term goes to zero, right? So we can just cross this out and rewrite our equation as tau dc dt equals f1 f dot h. So what is this? What does that look like? What's the solution to c for this differential equation? Does this exponentially relax toward a v infinity?

What is v infinity here? It's not even defined. If you set dc dt equal to zero, there's not even a c to solve for, right? So what is this? The derivative of c is just equal to-- if we put in an input that's constant, what is c?

AUDIENCE: [INAUDIBLE]

MICHALE FEE: This is an integrator, right? This c, the solution to this equation, is that c is the integral of this input. c is some initial c plus the integral over time. So if we have an input-- and again, what we're plotting here is the activity of one of the modes of our network, c1, which is a function of the projection of the input along the eigenvector of mode one.

So we're going to plot h, which is just how much the input overlaps with mode one. And as a function of time, let's start at one equals zero. What will this look like? This will just increase linearly. And then what happens? What happens here? Raymundo?

AUDIENCE: R just stays constant.

MICHALE FEE: Good. We've been through that, like, 100 times in this class. Now, what's special about this network is that remember, when lambda was less than one, the network would respond to the input. And then what would it do when we took the input

away? It would decay back to zero. But this network does something really special.

This network, you put an input in and then take the input away, this network stays active. It remembers what the input was. Whereas, if you have a network where lambda is less than one, the network very quickly forgets what the input was.

All right, what happens when lambda is greater than one? So when lambda is greater than one, this term is now-- this thing inside the parentheses is negative, multiplied by a negative number. This whole coefficient in front of the c1 becomes positive. So we're just going to write it as lambda minus one. And so this because positive. And what does that solution look like? Does anyone know what that looks like? dc dt equals a positive number times c.

Nobody? Are we all just sleepy? What happens? So if this is negative, if this coefficient were negative, dc-- if c is positive, then dc dt is negative, and it relaxes to zero, right? Lets think about this for a minute. What happens if this quantity is positive? So if c is positive-- cover that up.

If this is positive and c is positive, then dc dt is positive. So that means if c is positive, it just keeps getting bigger, right? And so what happens is you get exponential growth. So if we now take an input and we put it into this network, where lambda is greater than one, you get exponential growth.

And now what happens when you turn that input off? Does it go away? What happens? draw with their hand what happens here. So just look at the equation. Again, h dot f1 is zero here, so that's gone. This is positive. c is positive. So what is dc dt? Good. It's positive. And so what is--

AUDIENCE: [INAUDIBLE]

MICHALE FEE: It keeps growing. So you can see that this network also remembers that it had input. So this network also has a memory. So anytime you have lambda less than one the network just-- as soon as the input goes away, the network activity goes to zero, and it just completely forgets that it ever had input. Whereas, as long as lambda is equal to or greater than one, then this network remembers that it had input.

So if lambda is less than one, then the network relaxes exponentially back to zero after the input goes away. If you have lambda equal to one, you have an integrator,

and the network activity persists after the input goes away. And if you have exponential growth, the network activity also persists after the input goes away.

And so that right there is one of the best models for short-term memory in the brain. The idea that you have neurons that get input, become activated, and then hold that memory by reactivating themselves and holding their own activity high through recurrent excitation. But that excitation has to be big enough to either just barely maintain the activity or continue increasing their activity.

OK, now, that's not necessarily such a great model for a memory, right? Because we can't have neurons whose activity is exploding exponentially, right? So that's not so great. But it is quite commonly thought that in neural networks involved in memory, the lambda is actually greater than one.

And how would we rescue this situation? How would we save our network from having neurons that blow up exponentially? Well, remember, this was the solution for a network with linear neurons. But neurons in the brain are not really linear, are they? They have firing rates that saturate. At higher inputs, firing rates tend [AUDIO OUT]. Why? Because sodium channels become inactivated, and the neurons can't respond that fast, right?

All right, this I've already said. So we use what are called saturating non-linearities. So it's very common to write down models in which we can still have neurons that are-- we can still have them approximately linear. So it's quite often to have neurons that are linear for small [INAUDIBLE]. They can go plus and minus, but they saturate on the plus side or the minus.

So now you can have an input to a neuron that activates the neuron. You can see what happens is you start activating this neuron. It keeps activating itself, even as the input goes away. But now, what happens is that activity starts getting up into the regime where the neuron can't fire any faster. And so the activity becomes stable at some high value of firing. Does that make sense?

And this kind of neuron, for example, can remember a plus input, or it can remember a minus input. Does that make sense? So that's how we can build a simple network with a neuron that can remember its previous inputs with a lambda

that's greater than one. And this right here, that basic thing, is one of the models for how the hippocampus stores memories, that you have hippocampal neurons that connect to each other with a lot of recurrent connections [AUDIO OUT] in the hippocampus has a lot of recurrent connections.

And the idea is that those neurons activate each other, but then those neurons saturate so they can't fire anymore, and now you can have a stable memory of some prior input. And I think we should stop there. But there are other very interesting topics that we're going to get to on how these kind of networks can also make decisions and how they can store continuous memories-- not just discrete memories, plus or minus, on or off, but can store a value for a long period of time using this integrator. OK, so we'll stop there.