MICHALE FEE: Today, we're going to finish up with recurrent neural networks. So as you remember, we've been talking about the case where we have a layer of neurons in which we have recurrent connections between neurons in the output layer of our network. And we've been developing the mathematical tools to describe the behavior of these networks and describe how they respond to their inputs. And we've been talking about the different kinds of computations that recurrent neural networks can perform.

So you may recall that we started talking about-- we introduced the math or the concept of how to study recurrent neural networks by looking at the simplest recurrent network that has a single-- it's a single neuron with a recurrent connection called an autapse. A recurrent connection has a strength lambda. And we can write down-- let's see. So we can write down the equation for this, the response of this neuron, without a recurrent connection as tau dv dt equals minus v. The minus v is essentially a leak term, so that if you put input into the neuron, the response of the neuron jumps up and then decays exponentially in response to an input, h.

If we have a recurrent connection lambda, then there's an additional input to the neuron that's proportional to the firing rate of the neuron. We can rewrite that equation now as tau dv dt equals minus quantity one minus lambda times v plus the input. And the behavior of this simple recurrent neural network depends strongly on the value of this coefficient one minus lambda.

And we've talked about three different cases. We've talked about case where lambda is less than one, where lambda is equal to one-- in which case, this coefficient is zero-- and when lambda is greater than one. So let's look at those three cases again for this equation. So when lambda is less than one, you can see that this quantity right here, this coefficient in front of the v is negative. And what that means is that the firing rate of this neuron relaxes exponentially toward some h infinity-- sorry, some v infinity. And then when the input goes away, the neuron-- the firing rate decays exponentially towards zero.

OK, so in the case where lambda is equal to one, you can see that this coefficient is

zero. And now you can see that the derivative of the firing rate of the neuron is just equal to the input. What that means is that the firing rate of the neuron essentially integrates the input. And you can see, if you put a step input into this neuron with this recurrent connection of lambda equal one, that the response of the neuron simply ramps up linearly, which corresponds to integrating that step input.

And then when the input is turned off and goes back to zero, you can see that the firing rate of the neuron stays constant. And that's because the leak is exactly balanced by this excitatory recurrent input from the neuron onto itself.

So you can see that for the case for lambda equals one, there's persistent activity after you put an input into this neuron. And we talked about how this forms a short-term memory that can be used for a bunch of different things. It's a short-term memory of a scalar, or a continuous quantity, like I position. Or we talked about short-term memory integration being used for path integration or for accumulating evidence across noisy-- over long exposure to a noisy stimulus.

So today, we're going to focus on networks where this lambda is greater than one. And in that case, you can see that the differential equation looks like this. So if lambda is greater than one, then the quantity inside the parentheses here is negative. But that's multiplied by a minus one. So the coefficient in front of the v is positive. So if v itself is a positive number, then dv dt is also positive.

So if v is positive and dv dt is positive, then what that means is that the firing rate of that neuron is growing and, in this case, is growing exponentially. So that when you put an input in, the response of the neuron grows exponentially. But when you turn the input off, the firing rate of the neuron continues to grow exponentially, which is a little bit crazy. You know that neurons in the brain, of course, don't have firing rates that just keep growing exponentially.

So we're going to solve that problem by using nonlinearities in the firing F-I curve of neurons. But the key point here is that this kind of network actually remembers that there was an input, as opposed to this kind of network, where the when the input goes away, the activity of the network just decays back to zero. This kind of network has no memory that there was an input long ago in the past. Whereas, this kind of network remembers that there was an input. And so that kind of property when

lambda is greater than one is useful for storing memories.

So we're going to expand on that idea. In particular, we're going to use that theme to build networks that have attractors, that have stable states that they can go to, that depend on prior inputs, but also can be used to store long-term memories. All right? We're going to see how that kind of network can also be used to produce a winner-take-all network that is sensitive to which of two inputs are stronger and stores a memory of preceding inputs where one input is stronger than the other. Or it ends up in a different state when, let's say, input one is stronger than input 2, and it lands in a different state when input 2 is stronger than input one.

We're going to then describe a particular model, called a Hopfield model, for how attractor networks can store long-term memories. We're going to introduce the idea of an energy landscape, which is a property of networks that have symmetric connections, of which the Hopfield model is an example. And then we're going to end by talking about how many memories such a network can actually store, known as the capacity problem.

OK, so let's start with recurrent networks with lambda greater than one. So let's start with our autapse. Let's put lambda equal to 2. And again, you can see that if we rewrite this equation with lambda greater than one, we can write tau dv dt equals lambda minus one times v plus h. You can see that the value of zero, at the firing rate of zero, is an unstable fixed point of the network. Why is that? Because at v equals zero, then dv dt equals zero.

So what that means is that if the firing rate is exactly zero, that's a fixed point of the system. But if v deviates very slightly from zero, v becomes very slightly positive, then dv dt is positive, and the firing rate of the neuron starts running away. So what you can see is if you start the fire rate at zero and have the input at zero, then dv dt is zero, and the network will stay at zero firing rate.

But if you put in a very slight, a very small input, then dv dt goes positive, and the network activity runs away. Now, let's put in an input of the opposite sign. So now let's start with v equals zero and put in a very tiny negative input. What's the network going to do?

So tau dv dt equals v. So v is very slightly negative, or if h is very slightly negative

and v is zero, then dv dt will be negative, and the network will run away in the negative direction. So this network actually can produce two memories. It can produce a memory that a preceding input was positive, or it can store a memory that a preceding input was negative. So it has two configurations after you've put in an input that is positive or negative, right? It can produce a positive output or a negative output that's persistent for a long time. Yes?

AUDIENCE: Is the [INAUDIBLE] of a negative firing rate [INAUDIBLE]?

MICHALE FEE: Yeah. So you can basically reformulate everything that we've been talking about for neurons that have zero, that can't have negative firing rates. But in this case, we've been working with linear neurons. And it seems like the negative fire rates are pretty non-physical, non-intuitive. But it's a pretty standard way to do the mathematical analysis for neurons like this, is to treat them as linear.

But you can sort of reformulate all of these networks in a way that don't have that non-physical property. So for now, let's just bear with this slightly uncomfortable situation of having neurons that have negative firing rates. Generally, we're going to associate negative firing rates as inhibition, OK? But don't worry about that here.

All right, so we're going to solve this problem that these neurons have firing rates that are kind of running away exponentially by adding a nonlinear activation function. So a typical nonlinear activation function that you might use for linear neurons, like for networks of the type we've been considering, is a symmetric F-I curve, where if the input is positive and small, the firing rate of the neuron grows linearly, until you reach a point where it saturates. And larger inputs don't produce any larger firing rate of the neuron.

So most neurons actually have kind of a saturating F-I curve, like this, like the Hodgkin-Huxley neurons begin to saturate. Why is that? Because the sodium channels begin to inactivate, and it can't fire any faster than the-- there's a time between spikes that's sort of the closest that the neuron-- the fastest that the neuron can spike because of sodium channel inactivation.

And then on the minus side, if the input is small and negative, then the firing rate of the neuron goes negative linearly for a while and then saturates at some value. And we typically have the neuron saturating between one and minus one. So now, if you

start your neuron at zero firing rate and you put in a little positive input, what's the neuron going to do? Any guesses?

AUDIENCE: [INAUDIBLE]

MICHALE FEE: Yeah. It's going to start running up exponentially, but then it's going to saturate up here. And so the firing rate will run up and sit at one. And if we put in a negative input, a small negative input, then the neuron-- then this little recurrent network will go negative and saturate at minus one, OK?

So you can see that this network actually has one unstable fixed point, where if it sits exactly at zero, it will stay at zero, until you give a little bit of input in either direction. And then the network will run up and sit at another fixed point here of one. If you put in a big negative input, you can drive it to another fixed point. And these two are stable fixed points, because once they're in that state, if you give little perturbations to the network, it will deviate a little bit from that value.

If you give a small negative input, you can cause this to decrease a little bit. But then when the input goes away, it will relax back. So this is an unstable fixed point, and these are two stable fixed points.

Now, we're going to come back to this in more detail later. But we often think about networks like this as sort of like a ball on a hill. So you can imagine that you can describe this network using what's called an energy landscape, where if you start this system at some point on this sort of valley-shaped hill, all right, the network sort of-- it's like a ball that rolls downhill.

So if you start the network exactly at the peak, the ball will sit there. But if you give it a little bit of a nudge, it will roll downhill toward one of these stable points, OK? If you start it slightly on the other side, it will roll this way, OK? And those stable fixed points are called attractors. And this particular network has two tractors-- one with a firing rate of one and one at a firing rate of minus one. Yes, Appolonia?

AUDIENCE: The stable fixed points of the top graph, where'd you say they were?

MICHALE FEE: The stable fixed point is here, because once you-- if the system is in this state, you can give slight perturbations and the system returns to that fixed point. This is an

unstable fixed point, because if you start the system there and give it a little nudge in either direction, the state runs away. Does that makes sense?

AUDIENCE: Yeah.

MICHALE FEE: Any questions about that? Yes?

AUDIENCE: How is the shape of the curve [INAUDIBLE] points determined based on like--

MICHALE FEE: So I'm going to get-- I'm going to come back to how you actually calculate this energy landscape more formally. There's a very precise mathematical definition of how you define this energy landscape. All right, so this was all for the case of one neuron, all right? So now let's extend it to the case of multiple neurons.

So let's just take two neurons with an autapse. One of these autapses have a value strength of two, and the other autapse have a strength of minus two. So this one is recurrent and excitatory. This one is recurrent and inhibitory. So now what we're going to do is we can plot the state of the network.

Now, instead of being the state of the network in one dimension, v, we're now going to have v1 and v2. So the state of the system is going to be a point in a plane given by v1 and v2. So now, by looking at this network, you can see immediately that this particular neuron, this neuron with a firing rate of v2, looks like the kind of network that we've already studied, right? It has a stable fixed point at zero.

And this network has two stable fixed points-- one at one and the other one at minus one. So you can see that this system will also have two stable fixed points-- one there and one there, right? Because if I take the input away, this neuron is either going to one or minus one, and this neuron is going to go to zero. So there's one and minus one on the v1 axis. And those two states have zero firing rate on the v2 axis. Is that clear?

So now what's going to happen if we made this autapse have a strength of two? Anybody want to take a guess?

AUDIENCE: That's, like, four attractors?

MICHALE FEE: Right. Why is that?

AUDIENCE: Because that will also have stable fixed points at [INAUDIBLE].

MICHALE FEE: Right. So this one will have stable fixed points at one and minus one. This will also have stable fixed points at one and minus one, right? And the system can be in any one of four states-- 0, 0. Sorry, 1, 1; minus 1, minus 1; 1 minus 1; and minus 1, 1. That's right.

All right, so I just want to make one other point here, which is that no matter where you start the system for this network, it's going to evolve towards one of these stable fixed points, unless I started it exactly right there at zero. That will be another fixed point, but that's an unstable fixed point. OK, so this system will-- no matter where I start the state of that system, other than that exact point right there, the network will evolve toward one of those two attractors. That's why they're called attractors, because they attract the state of the system toward one of those two points. Yes?

AUDIENCE: So are the attractors determined by the nonlinear activation function?

MICHALE FEE: They are. So if this non-linear activation function saturated at two and minus two, then these two points would be up here at two and minus two. So you could see that this network has two eigenvalues, right? If we think of it as a linear network, this network has two eigenvalues. The connection matrix is given by a diagonal matrix with a two and a minus two along the diagonals, right?

So let's take a look at this kind of network. Now, instead of an autapse network, we have recurrent connections of strength minus 2 and minus 2. So what does that weight matrix look like?

AUDIENCE: 0, minus 2; minus 2, 0.

MICHALE FEE: 0, minus 2; minus 2, 0, right? Well, what are the eigenvalues of this network? Anybody remember that?

AUDIENCE: [INAUDIBLE]

MICHALE FEE: Right. It's a plus b and a minus b. And so the eigenvalues of this network are 0 plus negative 2 and 0 minus negative 2. So it's 2 and minus 2, right? So this network here will have exactly the same eigenvalues as this network. But what's going to be

different? What are the eigenvectors?

AUDIENCE: The 45.

MICHALE FEE: The 45 degrees. So the eigenvectors of this network are the x- and y-axes. The eigenvectors of this network are the 45-degree lines. So anybody want to take a guess as to what the stable states of this-- it's just this network rotated by 45 degrees, right? So those are now the attractors of this network, right?

And that makes sense, right? This neuron can be positive, but that's going to be strongly driving this neuron negative. But if this neuron is negative, that's going to be strongly driving this neuron positive, right? And so this network will want to sit out here on this line in this direction or in this direction. And because of the saturation-- if there were no saturation, if this were a linear network, the activity of this neuron would just be running exponentially up these 45-degree lines.

But because of the saturation, it gets stuck here at 1, minus 1. Or rather, minus 1, 1 or 1, minus 1. Any questions about that? Yeah, Jasmine?

AUDIENCE: So the two fixed points right now, like it's [INAUDIBLE]?

MICHALE FEE: Yeah. It'll be one in this direction and one in that direction.

AUDIENCE: So why [INAUDIBLE]?

MICHALE FEE: Because this neuron is saturated. Because the saturation is acting at the level of the individual neurons.

AUDIENCE: OK.

MICHALE FEE: So each neuron will go up to its own saturation point. OK? All right. So this kind of network is actually pretty cool. This network can implement decision-making. It can decide, for example, whether one input is bigger than the other, all right?

So if we have an input-- so let's start our network right here at this unstable fixed point, all right? We've carefully balanced the ball on top of the hill, and it just sits there. And now let's put an input that is in this direction h, so that it's slightly pointing to the right of this diagonal line. So what's going to happen? It's going to kick the state of the network up in this direction, right?

But we've already discussed how if the network state is anywhere on either side of that line, it will evolve toward the fixed point. If the h is on the other side, it will kick the network unstable fixed point into this part of the state space. And then the network will evolve toward this fixed point, OK?

These half planes here, this region here, is called the attractor basin for this attractor. And on this side, it's called attractor basin for that attractor, OK? And you can see that this network will be very sensitive to whichever input, h1 or h2, is slightly larger.

So let me show you what that looks like in this little movie. So we're going to start with our network exactly at the zero point. And we're going to give an input in this direction. And you can see that we've kicked the network slightly this way. And now the network evolves toward the fixed point, and it stays there. Now if we give a big input this way, we can push network over, push it to the other side of this dividing line between the two basins of attraction, and now the network sits here at this fixed point.

We can kick it again with another input and push it back. So it's kind of like a flip-flop, right? It's pretty cool. It detects which input was larger, pushes the network into an attractor that then remembers which input was larger for, basically, as long as the network-- as long as you allow the network to sit there. OK? All right, any questions about that? Yes, Rebecca?

AUDIENCE: Sorry. So the basin is just like each side of that [INAUDIBLE]?

MICHALE FEE: That's right. That's the basin of attraction for this attractor. If you start the network anywhere in this half plane, the network will evolve toward that attractor. And you can use that as a winner-take-all decision-making network by starting the network right there at zero. And small kicks in either direction will cause the network to relax into one of these attractors and maintain that memory.

Now let's talk about sort of a formal implementation of a system for producing memories, long-term memories, all right? And that's called a Hopfield model. And the Hopfield model is actually one of the best current models for understanding how memory systems like the hippocampus work.

So the basic idea is that we have neurons in the hippocampus, in particular in the CA3 region of the hippocampus, that have very prominent-- a lot of recurrent connectivity between those neurons, all right? And so you have input from entorhinal cortex and from the dentate gyrus that sort of serve as the stimuli that come into that network and form-- and burn memories into that part of the network by changing the synaptic weights within that network.

[INAUDIBLE] that some time later, when similar inputs come in, they can reactivate the memory in the hippocampus. And you recognize and remember that pattern of stimuli. All right, so we're going to-- actually, so an example of how this looks when you record neurons in the hippocampus, it looks like this.

So here's a mouse or a rat with electrodes in its hippocampus. If you put it in a little arena like this, it will run around and explore for a while. You can record where the rat is in that arena [AUDIO OUT] from neurons. And measure when the neurons spike and look at how the firing rate of those neurons relates to the position of the animal. So the black trace here shows all of the locations where the rat was when it was running around the maze, and the red dot shows where one of these neurons in CA3 of the hippocampus generated a spike, where the rat was when that neuron generates a spike. And those are shown with red dots here.

And you can see that this neuron generates spiking when the animal is in a particular restricted region of the cage, of its environment. And different neurons show different localized regions. So these regions are called place fields, because they are the places in the environment where that neurons spikes. Different neurons have different place fields. You can actually record from many of these neurons-- and looking at the pattern of neurons that are spiking, you can actually figure out where the rat was or is at any given moment, just by looking at which of these neurons is spiking.

That's pretty obvious, right? If this neuron is spiking and this neuron isn't, all these other neurons, then the animal is going to be-- you know that the animal is somewhere in that location right there. All right, so in a sense, the activity of these neurons reflects the animal remembering, or sort of remembering, that it's in a particular location. It's in a cage. It looks at the walls of the environment. It sees a

little-- they use colored cards on the wall to give the animal cues as to where it is. So they look around and they say, oh, yeah, I'm here. In my environment, there's a red card there and a yellow card there, and that's where I am right now.

So that's the way you think about these hippocampal place fields as being like a memory. On top of that, this part of the hippocampus is necessary for the actual formation of memories in a broader sense-- not just spatial locations, but more generally in terms of life events, right? For humans, the hippocampus is an essential part of the brain for storing memories.

All right, so let's come back to this idea of our recurrent network. And what we're going to do is we're going to start adding more and more neurons to our recurrent network. All right, so here's what the attractor looked like for the case where we have one eigenvalue in the system that's greater than one, another one that's less than one.

If we now make both of these neurons have recurrent connections that are stronger than one, now we're going to have four attractors, right? Each one of these has two stable fixed points-- a one and minus one. So here, for these two states, v1 is one. And for these two states, v1 is negative 1. For these two states, v2 is 1, and these two states, v2 is negative one, all right?

So you can see every time we add another neuron or another neuron to our network that has an autapse, every time we add another neuron with another eigenvalue, we add more possible states of the network, OK? So if we had two neurons, we have one neuron with an eigenvalue with an autapse greater than one, we have two states. If we have two, we have four states. If we have three of those, we have eight states.

So you can see that if we have n of these neurons with recurrent excitation with a lambda of greater than one, we have 2 to the n possible states that that system can be in, OK? So I don't know exactly how many neurons are in CA3. It has to be several million, maybe 10 million. We don't know the exact number. But 2 to that is a lot of possible states, right?

So the problem is that-- so let's think about how this thing acts as a memory. So it turns out that this little device that we've built here is actually a lot like a computer

memory. It's like a register, where we can write a value. So we can write in here a 1, minus 1, 1. And as long as we leave that network alone, it will store that value.

Or we can write a 1, 1, 1, and it will store that value. But that's not really what we mean when we talk about memories, right? We have a memory of meeting somebody for lunch yesterday, right? That is a particular configuration of sensory inputs that we experienced.

So the other way to think about this is this kind of network is just a short-term memory. We can program in some values-- 1, 1, 1. But if we were to turn the activity of these neurons off, we'd erase the memory, right? How do we build into this network a long-term memory, something that we can turn all these neurons off and then the network sort of goes back into the remembered state?

You do that by building connections between these neurons, such that only some of these possible states are actually stable states, all right? So let me give you an example of this. So if you have a whole bunch of neurons-- n neurons. You've got 2 to the n possible states that that network can sit in. What we want is for only some of those to actually be stable states of the system.

So, for example, when we wake up in the morning and we see the dresser or maybe the nightstand next to the bed, we want to remember that's our bedroom. We want that to be a particular configuration of inputs that we recall, right? So what you want is you want a set of neurons that have particular states that the system evolves toward that are stable states of the system.

So the way you do that is you take this network with recurrent autapses and you build cross-connections between them that make particular of those possible states actual stable states of the system. We want to restrict the number of stable states in the system.

So take a look at this network here. So here we have two neurons. You know that if you had autapses between these-- of these neurons to themselves, there would be four possible stable states. But if we now build excitatory cross-connections between those neurons, two of those states actually are no longer stable states. They become unstable. And only these two remain stable states of this system, remain attractors.

If we put inhibitory connections between those neurons, then we can make these two states the attractors of the system, OK? All right. Does that make sense?

All right, so let's actually flesh out the mathematics of how you take a network of neurons and program it to have particular states that are tractors of the system, all right? So we've been using this kind of dynamical equation. We're going to simplify that. We're going to follow the construction that John Hopfield used when he analyzed these recurrent networks.

And instead of writing down a continuous update so that we update the-- in the formulation we've been using, we update the firing rate of our neuron using this differential equation. We're going to simplify it by just writing down the state of the network at time t plus 1. That's a function of the state of the network of the previous time step. So we're going to discretize time.

We're going to say v, the state of the network, the firing rates of all the neurons at time t plus 1, is just a function of a weight matrix that connects all the neurons times the state of the system, times the firing rate vector. And then this can also have an input into it, all right? All right.

And here, I'm just writing out exactly what that matrix multiplication looks like. It's the state of the i-th [vector?] after we update the state of the network is just a sum over all of the different inputs coming from all of the other neurons, all the j other neurons. And we're going to simplify our neuronal activation function to just make it into a binary threshold neuron.

So if the input is positive, then the firing rate of neuron will be positive. If the input is negative, the firing rate of the neuron will be negative. All right? And that's the sine function. It's 1 if x is greater than 0 and minus 1 if x is less than or equal to 0. All right, so the goal is to build a network that can store any memory we want, any pattern we want, and turn that into a stable state.

So we're going to build a network that will evolve toward a particular pattern that we want. And xi is just a pattern of ones and minus ones that describes that memory that we're building into the network, all right? So xi is just a one or minus one for every neuron in the network. So xi i is one or minus one for the i-th neuron.

Now, we want xi to be an attractor, right? We want to build a network such that xi is an attractor. And what that means is that-- what does building a network mean? When we say build a network, what are we actually doing? What is it here that we're actually trying to decide?

AUDIENCE: The seminal roots.

MICHALE FEE: Yeah, which is?

AUDIENCE: Like the matrix M.

MICHALE FEE: The M, right. So when I say build a network that does this, I mean choose a set of M's that has this property. So what we want is we want to find a weight matrix M such that if the network is in a stable state, is in this desired state, that when we multiply that state times the matrix M and we take the sine of that sum, you're going to get the same state back.

In other words, you start the network in this state, it's going to end up in the same state. That's what it means to have an attractor, OK? That's what it means to say that it's a stable state. OK, so we're going to try a particular matrix. And I'm going to describe what this actually looks like in more detail. But the matrix that programs a pattern xi into the network as an attractor is this weight matrix right here.

So if we have a pattern xi, our weight matrix is some constant times the outer product of that pattern with itself. I'm going to explain what that means. What that means is that if neuron i and neuron j are both active in this pattern, both have a firing rate of one, then those two neurons are going to be connected to each other, right? They're going to have a connection between them that has a value of one, or alpha.

If one of those neurons has a firing rate of one and the other neuron has a firing rate of zero, then what weight do we want between them? If one of them has a firing rate of one and the other has a firing rate of minus one, the strength of the connection we want between them is minus one. So if one neuron is active and another neuron is active, we want them to excite each other to maintain that as a stable state.

If one neuron is plus and the other one is minus, we want them to inhibit each other, because that will make that configuration stable. OK, notice that's a symmetric matrix. So let's actually take our dynamical equation that says how we go from the state at time t to the state of time t plus 1 and put in this weight matrix and see whether this pattern xi is actually a stable state. So let's do that,

Let's take this M and stick it in there, substitute it in. Notice this is a sum over j, so we can pull the xi i out. And now, you see that v at t plus 1 is this. And it's the sine of a times xi i times the sum of j of xi j, xi k. Now, what is that? Any idea what that is?

So the elements of xi are what? They're just ones or minus ones. So xi j times xi j has to be?

AUDIENCE: One.

MICHALE FEE: One. And we're summing over n neurons. So this sum has to have a value N. So you can see that the state at time t plus 1-- if we start to network in this stored state, it's just this-- sine of a N xi. But a is positive. N is just a positive integer, number of neurons. So this equal xi.

So if we have this weight matrix, we start to network in that stored state, the state at the next time step will be the same state. So it's a stable fixed point. All right, so let's just go through an example. That is the prescription for programming a memory into a Hopfield network, OK? And notice that it's just-- it's essentially a Hebbian learning rule.

So the way you do this is you activate the neurons with a particular pattern, and any two neurons that are active together form a positive excitatory connection between them. Any two neurons where one is positive and the other is negative form a symmetric inhibitory connection, all right?

All right, so let's take a particular example. Let's make a three-neuron network that stores a pattern 1, 1, minus 1. And again, the notation here is xi, xi transpose. That's an outer product, just like you use to compute the covariance matrix of a data matrix.

So there's a pattern we're going to program in. The weight matrix is xi, xi transpose,

but it's 1, 1, minus 1 times 1, 1, minus 1. You can see that's going to give you this matrix here, all right? So that element there is 1 times 1. That element there. So here are two neurons.

These two neurons storing this pattern, these two neurons-- sorry, this neuron has a firing rate of minus one. So the connection between that neuron and itself is a one, right? It's just the product of that times that. All right any questions about how we got this weight matrix? I think it's pretty straightforward.

So is that a stable point? Let's just multiply it out. We take this vector and multiply it by this matrix. There's our stored pattern. There's our matrix that stores that pattern. And we're just going to multiply this out. You can see that 1 times 1 plus 1 times 1 plus minus 1 times minus 1 is 3. You just do that for each of the neurons.

Take the sine of that. And you can see that that's just 1, 1, minus 1. So 1, 1, minus 1 is a stable fixed point. Now let's see if it's actually an attractor. So when a state is an attractor, what that means is if we start to network at a state that's a little bit different from that and advance the network one time step, it will converge toward the attractor.

So into our network that stores this pattern 1, 1, minus 1, let's put in a different pattern and see what happens. So we're going to take that weight matrix, multiply it by this initial state, multiply it out, and you can see that next state is going to be the sine of 3, 3, minus 3. And one time step advanced, the network is now in the state that we've programmed in. Does that make sense?

So that state is a stable fixed point and it's an attractor. I'm just going to go through this very quickly. I'm just going to prove that xi is an attractor of the network if we write down the network as the outer product of this. The matrix elements are the outer product of the stored state, OK?

So what we're going to do is we're going to calculate the total input onto the i-th neuron if we start from an arbitrary state, v. So k is the input to all the neurons, right? And it's just that matrix times the initial state. So v j is the firing rate of the j-th neuron, and k is just M times v. That's the pattern of inputs to all of our neurons.

So what is that? k equals-- we're just going to put this weight matrix into this

equation, all right? We can pull the xi i outside of the sum, because it doesn't depend on j. The sum is over j. Now let's just write out this sum, OK? Now, you can see that if you start out with an initial state that has some number of neurons that have the correct sign that are already overlapping with the memorized state and some number of neurons in that initial state don't overlap with the memorized state, we can write out this sum as two terms.

We can write it as a sum over some of the neurons that are already in the correct state and a sum over neurons that are not in the correct state. So if these neurons in that initial state have the right sign, that means these two have the same sign. And so the sum over xi j vj for neurons where v has the right sign is just the number of neurons that has the correct sign. And this sum over incorrect neurons means these neurons have the opposite sign of the desired memory.

And so those will be one, and those will be minus one. Or those will be minus one, and those will be one. And so this will be minus the number of incorrect neurons. So you can see that the input of the neuron will have the right sign if the number of correct neurons is more than the number of incorrect neurons, all right?

So what that means is that if you program a pattern into this network and then I drive an input into the network, where most of the inputs drive-- if the input drives most of the neurons with the right sign, then the inputs will cause the network to evolve toward the memorized pattern in the next timestamp. OK, so let me say that again, because I felt like that didn't come out very clearly.

We program a pattern into our network. If we start to network at some-- let's say at zero. And then we put in a pattern into the network such that just the majority of the neurons are activated in a way that looks like the stored pattern, then in the next time step, all of the neurons will have this stored pattern. So let me show you what that looks like. Let me actually go ahead and show you-- OK, so here's an example of that.

So you can use Hopfield networks to store many different kinds of things, including images, all right? So this is a network where each pixel is being represented by a neuron in a Hopfield network. And a particular image was stored in that network by setting up the pattern of synaptic weights just using that xi, xi transpose learning

rule for the weight matrix M, OK?

Now, what you can do is you can [INAUDIBLE] that network from a random initial condition. And then let the network evolve over time, all right? And what you see is that the network converges toward the pattern that was stored in the synaptic [?], OK? Does that make sense? Got that?

So, basically, as long as that initial pattern has some overlap with the stored pattern, the network will evolve toward the stored pattern. All right, so let me define a little bit better what we mean by the energy landscape and how it's actually defined.

OK, so you remember that if we start our network in a particular pattern v, the recurrent connections will drive inputs into all the neurons in the network. And those inputs will then determine the pattern of activity at the next time step. So if we have a state of the network v, the inputs to the network, to all the neurons in the network, from the currently active neurons is given by the connection matrix times v. So we can just write that out as a sum like this.

So you define the energy of the network as the dot product-- basically, the amount of overlap-- between the current state of the network and the inputs to all of the neurons that drive the activity in the next step, OK? And the energy is minus, OK? So what that means is if the network is in a state that has a big overlap with the pattern of inputs to all the other neurons, then the energy will be very negative, right?

And remember, the system likes to evolve toward low energies. In physics, you have a ball on a hill. It rolls downhill, right, to lower gravitational energies. So you start the ball anywhere on the hill, and it will roll downhill. So these networks do the same thing. They evolve downward on this energy surface. They evolve towards states that have a high overlap with the inputs that drive the next state. Does that make sense?

So if you're in a state where the pattern right now has a high overlap with what the pattern is going to be in the next time step, then you're in an attractor, right? OK, so it looks like that. So this energy is just negative of the overlap of the current state of the network with the pattern of inputs to all the neurons. Yes, Rebecca?

AUDIENCE: So [INAUDIBLE] to say [INAUDIBLE] with the weight matrix, since that's sort of the

goal of the next time step, and it will evolve towards the matrix [INAUDIBLE]?

MICHALE FEE: Yeah. So the only difference is that the state of the network is this vector, right? And the weight matrix tells us how that state will drive input into all the other neurons. And so if you're in a state that drives a pattern of inputs to all the neurons that looks exactly like the current state, then you're going to stay in that state, right? And so the energy is just defined as that dot product, the overlap of the current state, or the state that you're calculating the energy of, and the inputs to the network in the next time step.

All right, so let me show you what that looks like. And so the energy is lowest, current state has a high overlap with the synaptic drive to the next step. So let's just take a look at this particular network here. I've rewritten this dot product as-- so k is just M times v. This dot product can just be written as v transpose times Mv. So that's the energy.

Let's take a look at this matrix, this network here-- 0, minus 2, minus 2, 0. So it's this mutually inhibitory network. You know that that inhibitory network has attractors that are here at minus 1, 1 and 1, minus 1. So let's actually calculate the energy.

So you can actually take these states-- 1, minus 1-- multiply it by that M, and then take the dot product with 1, minus 1. And do that for each one of those states and write down the energy. You can see that the energy here is minus 1. The energy here is minus 1, and the energy here is 0. So if you start the network here, at an energy zero, it's going to roll downhill to this state. Or it can roll downhill to this state, depending on the initial condition, OK?

So you can also think about the energy as a function of firing rates continuously. You can calculate that energy, not just for these points on this grid. And what you see is that there's basically-- in high dimensions, there are sort of valleys that describe the attractor basin of these different attractors, all right? And if you project that energy along an axis like this, you can see that you sort of-- let's say, take a slice through this energy function. You can see that this looks just like the energy surface, the energy function, that we described before for the 1D factor, the single neuron with two attractors, right?

This corresponds to a valley and a valley and a peak between them. And then the

energy gets big outside of that. And questions about that? Yes, [INAUDIBLE].

AUDIENCE: [INAUDIBLE] vector 1/2 because-- in this case, right?

MICHALE FEE: That's the general definition, minus 1/2 v dot k. It actually doesn't really-- this 1/2 doesn't really matter. It actually comes out of the derivative of something, as I recall. But a scaling factor doesn't matter. The network always evolves toward a minimum of the energy. And so this 1/2 could be anything.

All right, so the point is that starting the network anywhere with a sensory input, the system will evolve toward the nearest memory, OK? And I already showed you this. OK, so now, a very interesting question is, how many memories can you actually store in a network? And there's a very simple way of calculating the capacity of the Hopfield network.

And I'm just going to show you the outlines of it. And that actually gives us some insight into what kinds of memories you can store. Basically, the idea is that when you store memories in a network, you want the different memories to be as uncorrelated with each other as possible. You don't want to try to store memories that are very similar to each other. And you'll see why in a second when we look at the map.

So let's say that we want to store multiple memories in our network. So instead of just storing one pattern, xi, we want to store a bunch of different patterns xi. And so let's say we're going to store P different patterns. So we have a parameter variable mu. An index mu addresses each of the different patterns we want to store. So we're going to store zero to p patterns, p minus 1 patterns.

So what we do, the way we do that is we compute the contribution to the weight make matrix from each of those different patterns. So we calculate a weight matrix using the outer product for each of the patterns we want to store in the network, all right? And then we add all of those together. We're going to essentially sort of average together the network that we would make for each pattern separately. Does that makes sense?

So there is the equation for the weight matrix that stores p different patterns in our memory, in our network. And that's how we got this kind of network here, where we

store multiple memories, all right? So let me just show you an example of what happens when you do that. So I found these nice videos online.

So here is a representation of a network that stores a five by five array of pixels. And this network was trained on these three different patterns. And what this little demo shows is that if you start the network from different configurations here and then evolve the network-- you start running it. That means you run the dynamic update for each neuron one at a time, and you can see how this system evolves over time.

So this is a little GUI-based thing. You can flip the state and then run it. And you can see that if you change those, now it-- I think he was trying to make it look like that. But when you run it, it actually evolved toward this one. He's going to really make it look like that. And you can see it evolves toward that one.

All right, any questions about that? You can see it stored three separate memories. You've given an input, and the network evolves toward whatever memory was closest to the input. So that's called a content [INAUDIBLE] memory. You can actually recall a memory-- not by pointing to an address, like you do in a computer, but by putting in something that looks a little bit like the memory. And then the system evolves right to the memory that was closest to the input.

So it's also called an auto-associative memory. It automatically associates with the nearest-- with a pattern that's nearest to the input. So here's another example. It's just kind of more of the same. This is a network similar to this. Instead of black and white, it's red and purple, but it's got a lot more pixels. And you'll see the three different images that are stored in there-- so a face, a world, and a penguin.

So then what they're doing here is they add noise. And then you run the network, and it [AUDIO OUT] one of the patterns that you stored in it. So here's the penguin. Add noise. Add a little bit of noise. Here, he's coloring it in, I guess, to make it. And then you run the network, and it remembers the [AUDIO OUT].

OK, so that's interesting. So he ran it. He or she ran the network. And you see that it kind of recovered a face, but there's some penguin head stuck on top. So what goes wrong there? Something bad happened, right? The network was trained with a face, a globe, and a penguin. And you run it most of the time, and it works. And then,

suddenly, you run it, and it recovers a face with a penguin head sticking out of it. What happened?

So we'll explain what happens. What happened was that that this network was trained in a way that has what's called a spurious attractor. And that often happens when you train a network with too many memories, when you exceed the capacity of the network to store memories. So let me show you what actually goes wrong mathematically there.

All right, so we're going to do the same analysis we did before. We're going to take a matrix. We're going to build a network that stores multiple memories. This was the matrix to build one memory. Let's see what I did here. So in order for-- Yeah. Sorry.

This was the matrix for multiple memories. We're summing mu. I just didn't write the mu equals 0 to p. So we're going to program p different memories by summing up this outer product for all the different patterns that we're wanting to store, all right? We're going to ask whether one of those-- under what conditions is one of those patterns, the xi 0, actually a stable state of the network?

So we're going to build a network with multiple patterns stored, and we're just going to ask a simple question. Under what conditions is xi 0 going to evolve to xi 0? And if xi 0 evolves toward xi 0, or stays at xi 0, then it's a stable point. All right, so let's do that.

We're going to take that weight matrix, and we're going to plug-in our multiple memory weight matrix, all right? You can see that we can pull out the xi i out of this sum over j. And the next step is we're going to separate this into a sum over mu equals zero and a separate sum for mu not equal to 0, all right? So this is a sum over all the mu's, but we're going to pull out the mu zero term as a separate sum over j. Is that clear?

Anyway, this is just for fun. You don't have to reproduce this, so don't worry. So we're going to pull out the mu equals zero term. And what does that look like? It's xi i0, sum over j of xi j0, xi j0. So what is that? That's just N, right, the number of neurons. We're summing over j equals 1 to N, number of neurons. I should add those limits here.

So you can see that that's N. So this is just sine of xi i0 plus a bunch of other stuff. So you can see right away that if all of this other stuff is really small, then this is a fixed point. Because if all this stuff is small, the system will evolve toward the sine of xi [INAUDIBLE], which is just xi i0.

So let's take a look at all of this stuff and see what can go wrong to make this not small. All right, so let's zoom in on this particular term right here. So what is this? This is sum over j, xi mu j, xi 0 j. So what is that? Anybody know what that is? It's a vector operation. What is that?

AUDIENCE: The dot product between one image and then zero.

MICHALE FEE: Exactly. It's a dot product between the image that we're asking is it a stable fixed point and all the other images in the network. Sorry, and the mu-th image. So what this is saying is that if our image is orthogonal to all the other images in the network that we've tried to store, then this thing is zero.

So this is referred to as crosstalk between the stored memories. So if our pattern, xi 0, is orthogonal to all the other patterns, then it will be a fixed point. So the capacity of the network, the crosstalk-- the capacity of the network depends on how much overlap there is between our stored pattern and all the other patterns in the network, all right?

So if all the memories are orthogonal, if all the patterns are orthogonal, then they're all stable attractors. But if one of those memories, xi 1-- let's take xi 1-- is close to xi 0, then xi 0 dot xi 1-- the two patterns are very similar-- then the dot product is going to be N, right? And when you plug that, if that's N, then you can see that this becomes xi 1 i, right?

So what happens is that these other memories that are similar to our memorized pattern-- then when you sum that, when you compute that sum, some of these terms get big enough so that the memory in the next step is not that stored memory. It's a combination. All right?

So what happens is-- so the way the capacity of the network is stored. So you can't actually choose all your memories to be orthogonal. But a pretty good way of making memory is nearly orthogonal is to store them as random [AUDIO OUT]. So a

lot of the thinking that goes into how you would build a network that stores a lot of patterns is to take your memories and sort of convert them in a way that makes them maximally orthogonal to each other. You can use things like lateral inhibition to orthogonalize different inputs.

So once you make your patterns sort of noisy, then it turns out you can actually calculate that if the values of xi sort of look like random numbers, that you can store up to about 15% of the number of neurons worth of memories in your network. So if I have 100 neurons in my network, I should be able to store about 15 different states in that network before they start to interfere with each other, before you have a sufficiently high probability that two of those memories are next to each other.

And as soon as that happens, then you start getting crosstalk between those memories that causes the state of the system to evolve in a way that doesn't recall one of your stored memories, all right? And what that looks like in the energy landscape is when you build a network with, let's say, five memories, there will be five minima in the network that sort of have equal low values of energy.

But when you start sticking too many memories in your network, you end up with what are called spurious attractors, sort of local minima that aren't at the-- that don't correspond to one of the stored memories. And so as the system evolves, it can be going downhill and get stuck in one of those minima that look like a combination of two of the stored memories. And that's what went wrong here with the guy with the penguin sticking out of his head.

Who knows? Maybe that's what happens when you look at something and you're confused about what youre seeing. We don't know if that's actually what happens, but it would be an interesting thing to test. Any questions?

All right, so that's-- so you can see that these are long-term memories. These don't depend on activity in the network to store, right? Those are programmed into the synaptic connections between the neurons. So you can shut off all the activity. And if you just put in up a pattern of input that reminds you of something, the network will recover the full memory for you.