MICHALE FEE: So for the next few lectures, we're going to be looking at developing methods of studying the computational properties of networks of neurons. This is the outline for the next few lectures.

Today we are going to introduce a method of studying networks called a rate model where we basically replace spike trains with firing rates in order to develop simple mathematical descriptions of neural networks. And we're going to start by introducing that technique to the problem of studying feed-forward neural networks. And we'll introduce the idea of perceptrons trance as a method of developing networks that can classify their inputs.

Then in the next lecture, we're going to turn to largely describing mathematical tools based on matrix operations and the idea of basis sets. Matrix operations are very important for studying neural networks. But they're also a fundamental tool for analyzing data and doing things like reducing the dimensionality of high dimensional data sets, including methods such as principal components analysis. So it's a very powerful set of methods that apply both to studying the brain and to analyzing the data that we get when we study the brain.

And then finally we'll turn to a few lectures that focus on recurrent neural networks. These are networks where the neurons connect to each other densely in a recurrent way, meaning a neuron will connect to another neuron. And that neuron will connect back to the first neuron.

And networks that have that property have very interesting computational abilities. And we're going to study that in the context of line attractors and short-term memory and hopfield networks.

So for today, the plan is to develop the rate model. We're going to show how we can build receptive fields with feed forward networks that we've described with the rate model. We're going to take a little detour and describe vector notation and vector algebra, which is very important for these models, and also for building up to the matrix methods that we'll talk about in the next lecture. Again, we'll talk about neural networks for classification and introduce the idea of a perceptron. So that's

for today.

So I've already talked about most of this. Why is it that we want to develop a simplified mathematical model of neurons that we can study analytically? Well, the reason is that we can really develop our intuition about how networks work.

And that intuition applies not just to the very simplified mathematical model that we're developing, but also applies more broadly to real networks with real neurons that actually generate spikes and interact with each other by the more complex biophysical mechanisms that are going on in the brain.

So a good example of this is how we simplified the detailed spiking neurons of the Hodgkin-Huxley model and approximate that as an integrate and fire model, which captures a lot of the properties of real neurons. Simplifies it enough to develop an intuition, but captures a lot of the important properties of real neural circuits.

All right, so let's start by developing the basic idea of a rate model. Let's start with two neurons. We have an input neuron and an output neuron. The input neuron has some firing rate given by u. And the output neuron has some firing rate given by v. So we're going to essentially ignore the times of the spikes and describe the inputs and outputs of this network just with firing rates. You can think of the rate as just having units have spikes per second, for example.

Those neurons, the input neuron and the output neuron, are connected to each other by a synapse. And we're going to replace all of the complex structure of synapses, vesicle release, neurotransmitter receptors, long-term depression and paired spike facilitation and depression, all that stuff we're just going to ignore. And we're going to replace that synapse with a synaptic weight w.

Just to give you the simplest intuition of how a rate model works, there are models where we can just treat the firing rate of the output neuron, for example, as linear in its input. And we can simplify this even to the point where we can describe the firing rate of the output neuron as the synaptic weight w times the firing rate of the input neuron. So that's just to give you a flavor of where we're heading.

And I'm going to justify how we can do this and/or why we can do this. And then we're going to build this up from the case of one input neuron and one output

neuron to the case where we can have many input neurons and many output neurons.

So how do we justify going from spikes to firing rates? So remember that the response of a real output neuron, a real neuron, to a single spike at its input, is some change in the postsynaptic conductance that follows an input spike. And in our model of a synapse, we described that the input spike produces a transient increase in the synaptic conductance. And that synaptic conductance we modeled as a simple step increase in the conductance followed by an exponential decay as the neurotransmitter gradually unbinds from the neurotransmitter receptors.

So we have a transient change in the synaptic conductance. That's just a maximum conductance times an exponential decay. Now remember that we wrote down the postsynaptic-- we can write down the postsynaptic current that results from this synaptic input as the synaptic conductance times v minus e synapse, the synaptic reversal potential.

In moving forward in this model, we're not going to worry about synaptic saturation. So we're just going to imagine that the synaptic current is just proportional to the synaptic conductance.

So now we can write the conductance as just some weight times a kernel that is just some kernel of unit area. So what we've done here is we've just taken the synaptic current and we've written it as a constant, a synaptic weight, times an exponentially decaying kernel of area, area 1.

So now if we have a train of spikes at the input instead of a single spike, we can write down that train of spikes, the spike train, as a sum of delta functions where the spike times are t sub i. And if you want to plot the synaptic current as a function of time, you would just take that spike train input and do what with that linear kernel? We would convolve it, right?

So we would take that spike train, convolve it with that little exponential kernel. And that would give us the synaptic current that results from that spike train.

So let's think for a moment about what this quantity is right here. What is k, this k which is a little kernel that has an exponential step, and then an exponential decay?

What do you get when you convolve that kind of smooth kernel with this spike train here? What does that look like?

We did that at one point when we were in class when we were talking about how you would estimate something from a spike train. What is that? What is that quantity right there? It's sort of a smoothed version of a spike train, which is how you would calculate what, Habiba?

AUDIENCE: Is it a window for the spike train?

MICHALE FEE: Yeah. It's windowed, but what is it that you are calculating when you take a spike train and you convolve it with some smooth window?

AUDIENCE: Low-pass window?

MICHALE FEE: It's like a low-pass version of the spike train. And remember in the lecture on firing rates, we talked about how that's a good way to get a time-dependent estimate of the firing rate of a neuron. We take the spike train and just convolve it with a smooth window.

And if the area of that smooth window is 1, then what we're doing is we're estimating the firing rate of the neuron as a function of time. Does that make sense? Yes?

AUDIENCE: So k is just a kernel?

MICHALE FEE: k is just is smooth kernel that happens to have this exponential shape.

AUDIENCE: Is it like [INAUDIBLE]

MICHALE FEE: Well, that's our model for how a synapse-- basically, what I'm saying is that when you take a spike train and put it through a synapse, what comes out the other end is a smoothed version of the spike train.

AUDIENCE: OK.

MICHALE FEE: That's all this is saying.

AUDIENCE: OK. [INAUDIBLE] they have this area or quantity?

MICHALE FEE: Yep. If k has-- you remember that if k has an area 1, then when you convolve evolve that kernel with the spike train, you get a number that has units of spikes per second. And that quantity is an estimate of the local firing rate of the neuron. Does that make sense?

So basically, we can take this spike train, and by convolve it with a smooth window, we can estimate the number of spikes per second in that window. So what do we have here? We have that the current is just a constant times an estimate of the firing rate at that time. If k is a kernel, a smooth kernel with an area normalized to 1, then this quantity is just an estimate of the firing rate.

So let's take a look at that. So here I have just made a sample spike train with a bunch of spikes that look like they're increasing in firing rate and decreasing in firing rate. If we take that spike train and convolve it with this kernel, you can see that you get this sort of broad bump that looks like it gets higher in the middle where the firing rate is higher. And it's lower at the edges where the firing rate is lower.

So the point is that you can take a spike train and put it into a neuron. The response of the neuron is a smooth low-pass version of the rate of this input spike train. And so you can think about writing down the input to this neuron as a weight times the firing rate of the input.

So that was a way of writing down the input to this output neuron from the input neuron, the current input. Now what is the firing rate of the output neuron in response to that current injection? So that's what we're going to ask next.

And you can remember that when we talked about the integrate and fire model, we saw that neurons in the approximation of large inputs have firing rate as a function of current that looks like this. It's zero for inputs below the threshold current. For input currents that aren't large enough to drive the neuron to threshold, the neuron doesn't spike at all.

And then above some threshold, the neuron fires approximately linearly at higher input currents. So the way that we think about this is that the input on is spiking at some rate. It goes through a synapse. That synapse smooths the input and produces some current in the postsynaptic neuron that's proportional

approximately to the firing rate of the input neuron. And the output neuron has some output firing rate that's some function of the input current.

So we can write down the firing rate of our output neuron, v. It's just equal to some function of the input current, which is just some function of w times the firing rate of the input neuron. And that right there is the basic equation of the rate model. The output firing rate is some function of a weight times the firing rate of the input neuron.

And everything else about the rate model is just different rate models have different numbers of input neurons where we have more than one contribution to the input current. They can have many output neurons. They can have different FI curves for the output neurons.

Some of them are non-linear like this. Some of them are linear. And we're going to come back and talk about the function of different FI curves and why different FYI curves are useful. Any questions about this? That's the basic idea. All right, good.

So let's take one particularly simple version of the rate model called a linear rate model. And the linear rate model has a particular FI curve. That FI curve says that the firing rate of the neuron is linear in the input current.

Now why is this a really weird model of a neuron? What's fundamentally non-biological about this?

AUDIENCE: Negative firing rate.

MICHALE FEE: I'm hearing a bunch of right answers at the same time.

AUDIENCE: Negative firing rate.

MICHALE FEE: This neuron is allowed to fire at a negative firing rate if the input current is negative. That's a pretty crazy thing to do. Why do you think we would want to do that?

AUDIENCE: [INAUDIBLE]?

MICHALE FEE: Well, no actually we do. So you can have inhibitory inputs that produce outward currents that hyperpolarize the neuron. Any thoughts about that?

It turns out that as soon as you have your output neurons have this kind of FI curve, a linear FI curve, then the math becomes super simple. You can write down very complex networks of neurons with a bunch of linear differential equations. And it becomes very easy to write down what the solution is to how a network behaves as a function of its inputs.

And we're going to spend a lot of time working with network models that have linear FI curves because you can develop a lot of intuition about how networks behave by using models like this. As soon as you have models like this, you can't solve the behavior of the network analytically. You have to do everything on the computer. And it becomes very hard to derive general solutions for how things behave. So we're going to use this model a lot.

And in this case again, for the case of this two-neuron network where we have one output neuron that receives a synaptic input from an input neuron, the firing rate of the output neuron is just w, the synaptic weight times the firing rate of the input neuron. And we're going to come back to non-linear neurons because that non-linearity actually does really important things. And we're going to talk about what that does.

So now let's look at the case where our output neuron has not just one input but actually many inputs from a bunch of input neurons. So here we have what we call an input layer, a layer of neurons in the input layer. Each one of those neurons has a firing rate-- u1, u2, u3, u4, u5.

Each of those neurons sends a synapse onto our output neuron. Each one of those synapses has a synaptic weight. This weight is w1. And that's w2, w3, w4, and w5.

Now you can see that the total input, the total current, to this output neuron is just going to be a sum of the inputs from each of the input neurons. The total input is just a sum of the inputs from each of the input neuron. So the synaptic current-- total synaptic current into this neuron is w1 times u1, plus w2 times u2, plus w3 times u3, plus all the rest.

So the response of our linear neuron, the firing rate of our linear neuron, is just a sum over all of those inputs. So again, in this case, we're going to say that the total

input current to this neuron is the sum over this. But then because this is a linear neuron, the firing rate is just equal to that current input. Does that make sense?

So you can see that this description of the firing rate of the output neuron is a sum over all of those contributions. It turns out that this actually can be written in a much more compact way in vector notation. What does that look like? Does anyone know in vector notation what that looks like?

AUDIENCE: Dot product.

MICHALE FEE: That's a dot product. That's right. So in general, it's much easier to write these responses in vector notation. And so I'm just going to walk you through some basics of vector notation for those of you who might need a few minutes of reminder.

Actually before we get to the vector notation, I just want to describe how we can use a simple network like this to build a receptive field. So you remember that when we were talking about receptive fields of neurons, we described how a neuron can have a maximal response to a particular pattern of input. So let's say we have a neuron that's sensitive to visual inputs. And as a function of one dimension, let's say along the retina, this neuron has a big response if light comes in central field, some inhibitory responsive light comes in outside of that central lobe.

Well, it turns out that a very simple way to build neurons that have receptive fields like this, for example, is to have an input layer that projects to this neuron that has this receptive field and has a pattern of synaptic inputs that corresponds to that pattern in the field. So you can see that if this neuron-- so let's say these are neurons in the retina, let's say retinal ganglion cells, and this neuron is in the thalamus, we can build a thalamic neuron that has a center-surround receptive field like this by having let's say this neuron has a strong positive excitatory synaptic weight onto our output neuron.

So you can see that if you have light here that corresponds to this neuron having a high firing rate, that neuron is very effective at driving the output neuron. And so the output neuron has a positive component of its receptor field right there in the middle.

Now if this neuron here, which is in this part of the retina, if that neuron has a

negative weight onto the output neuron, then light coming in here driving this neuron will inhibit the output neuron. So if you have a pattern of weights that looks like this, 0 minus 1, 2 minus 1, 0, that this neuron will have a receptive field that looks like that as a function of its inputs.

So that's a on-dimensional example. And you can see that you write down the output here as a weighted sum of each one of those inputs.

This also works for two dimensional receptive fields. For example, if we have input from the retina that looks like this where we have-- I guess this was excitatory here in the center, inhibitory around, you can make a neuron that has a two-dimensional receptor field like this by having inputs to this neuron from all of those different regions of the visual field that have different weights corresponding to positive in the center. So neurons in the positive synaptic weights under the output neuron. And neurons around the edges have negative synaptic weights.

So we can build any receptive field we want into a neuron by just plugging in-- by putting in the right set of synaptic weights. Yes?

AUDIENCE: So would you rule out [INAUDIBLE]

MICHALE FEE: So in real life, I assume you mean in the brain?

AUDIENCE: Yeah.

MICHALE FEE: So in the brain, we don't really know how these weights are built. So one idea is that there are rules that control the development of these circuits, let's say, connections of bipolar cells in the retina to retinal ganglion cells that control how these weights are determined to be positive or negative.

Negative weights are implemented by bipolar cells connected to amacrine cells, which are inhibitory, and then connect to the retinal ganglion. So there's a whole circuit that gets built in the retina that controls whether these weights are positive or negative. And those can be programmed by genetic developmental programs. They can also be controlled by experience with visual stimuli.

So there's a lot we don't understand about how these weights are controlled or set up or programmed. But the way we think about how receptive fields of these

neurons emerge is by controlling the weight of those synaptic input. That's the message here-- that receptive fields emerge from the pattern of weights from an input layer onto an output layer.

AUDIENCE: [INAUDIBLE] how many [INAUDIBLE]

MICHALE FEE: If you're going to build a model, let's say, of the retina. So it just depends on how realistic you want it to be. If you wanted to make a model of a retinal ganglion cell, you could try to build a model that has as many bipolar neurons as are actually in the receptive field of that retinal ganglion cell.

Or you could make a simplified model that only has 10 or 100 neurons. Depends on what you want to study. All right any other questions?

And again, even for these more complex models, you can still write down a simple rate model formulation of the firing rate of the output neuron. It's just a weighted sum of the input firing rate.

So each neuron in the input layer fires at some rate. It has a weight w. To get the contribution of this neuron to the firing rate of the output neuron, you just take that input firing rate times the synaptic weight, and add that up then for all the input layer neurons.

So as I said, we've been describing the response of our linear neuron as this weighted sum. And that's a little bit cumbersome to carry around. So we're going to start using vector notation and matrix notation to describe networks. It's just much more compact.

So we're going to take a little detour, talk about vectors. So a vector is just a collection of numbers. The number of numbers is called the dimensionality of the vector. If a vector has only two numbers, then we can just plot that vector in a plane.

So for a 2D vector, if that vector has two components, x1 and x2, then we can plot that vector in that space of x1 and x2, put the origin at zero. In this case, the vector has two vector components or elements, x1 and x2.

And in two dimensions we describe that as spaces, as R2, the space of two real

numbers. We can write down that vector as a row in row vector notation. So x is x1, x2. We can write it as a column vector, x1, x2, organized on top of each other, like this.

Vector sums are very simple. So if you have two vectors, x and y, you can write down the sum of x and y is x plus y. That's called the resultant. X plus y it can be written like this in column vector notation.

You can see that the sum of x and y is just an element by element sum of the vector elements. It's called element by element addition. Let's look at vector product.

So there are multiple ways of taking the product of two vectors. There's an element by element product, an inner product, an outer product that we'll cover in later lectures. And also, something called the cross product that's very common in physics.

But I have not yet seen the application of a cross product to neuroscience. If anybody can find one of those, I'll give extra credit.

Element by element product is called a Hadamard product. So x times y is just the element-by-element product of the elements in the two vectors. In Matlab, that element-by-element product you compute by x dot star y.

Inner product or dot product looks like this. So if we have two column vectors, the dot product of x and y is the sum of the element-by-element products. So x dot y is just x1 times y1 plus x2 times y2, and so on, plus xn times yn. And that's that sum that we saw earlier in our feed forward network.

OK. So notice that the dot product is a scalar. It's a single number. It's no longer a vector. Products have some nice properties. They're commutative.

So x.y is equal to y.x. They're distributive so that vector w dotted into the sum of two vectors is just the sum of the two separate dot products. So w dot x plus y is just w.x, w.y. And it's also linear.

So if you have a x dot y that is equal to a times the quantity x.y. So if you have vector x and y dotted into each other, if you make one of those vectors twice as long, then the dot product is just twice as big.

A little bit more about inner products. So we can also write down the inner product in matrix notation. So x.y is a matrix product of a row vector.

Column vector, you remember how to multiply two matrices. You multiply the elements of each row times the elements of each column. So you can see that this in matrix notation is just the dot product of those two vectors.

In matrix notation, this is a 1 by n matrix. This is an n by 1. So 1 row by n columns, times n rows by 1 column. And that is equal to a 1 by 1 matrix, which is just a scalar.

All right, in Matlab, let me just show you how to write down these components. So in this case, x is a column vector, a 1 by 3 column vector. y is a 1 by 3 column vector. You can calculate those vectors like this. And z is x transpose times y. And so that's how you can write down the dot product of two vectors.

What is the dot product of a vector with itself? It's the square magnitude of the vector. So x is just the norm or magnitude of the vector. And you can see that the norm of the vector is just-- you can think about this as being analogous to the Pythagorean theorem. The length of one side of a triangle is just the sum of the squares of all the sides, the square root of that.

So a unit vector is a vector that has length 1. So a unit vector by definition has a magnitude of 1, which means its dot product with itself is 1.

We can turn any vector into a unit vector by just taking that vector, dividing by its norm. I'm going to always use this notation with this little caret symbol to represent a unit vector. So if you see a vector with that little hat on it, that means it's a unit vector.

You can express any vector as a product of a scalar, a length, times a unit vector in that direction. We can find the projection or component of any vector in the direction of this unit vector as follows. So if we have a unit vector x, we can find the projection of a vector y onto that unit vector x.

How do we do that? We just find the normal projection of that vector. That distance right there is called the scalar projection of y onto x. If you write down the length of the vector y, the norm of the vector y in the angle between y and x, then the dot

product y.x is just equal to the magnitude of y times the cosine of the angle between the two vectors. Just simple trigonometry.

We can also define what's called the vector projection of y onto x as follows. So we just draw that same picture. So we can find the projection of y onto x and add that as a vector.

And that's just this scalar projection of y onto x times a unit vector in the x direction. So x actually is a unit vector in this example. So this vector projection of y to x is just defined as y dot x times x. Any questions about that?

I'm guessing most of you have seen all of this stuff already. But we're going to be using these things a lot. So I just want to make sure that we're all on the same page. And that's just a scalar times a unit vector.

Let me just give you a little bit of intuition about dot products here. So a dot product is related to the cosine of the angle between two vectors, as we talked about before. The dot product is just magnitude of x times the magnitude of y times the cosine of the angle between them.

So the cosine of the angle between two vectors is just the dot product divided by the product of the magnitude of each of the two vectors. So if x and y are unit vectors, the cosine of the angle between them is just the dot product of the unit vectors. So again, if x and y are unit vectors, then that dot product is just the cosine of the angle.

Orthogonality. So two vectors are orthogonal, are perpendicular, if and only if their dot product is 0. So if we have two vectors x and y, they are orthogonal if the angle between them is 90 degrees. x.y is just proportional to the cosine of the angle. Cosine of 90 degrees is zero.

So if two vectors are orthogonal, then their dot product will be zero. If their dot product is zero, then they're orthogonal with each other. And using the notation we just developed, the vector projection of y along x is the zero vector, if those two vectors are orthogonal.

There is an intuition that one can think about in terms of the relation between dot product and correlation. So the dot product is related to the statistical correlation

between the elements of those two vectors. So if you have a vector x and y, you can write down the cosine of the angle between those two vectors, again, as x.y over the product of the norms.

And if you write that out as sums, you can see that this is just the sum of the element-by-element products-- that's the dot product-- divided by the norm of x and the norm of y. And if you have taken a statistics class, you will recognize that as just the Pearson correlation of a set of numbers x and a set of numbers y. The dot product is closely related to the correlation between two sets of numbers.

One other thing that I want to point out coming back to the idea of using this feed forward network as a way of receptive field, you can see that the response of a neuron in this model is just the dot product of the stimulus vector u. The vector of input firing rates represents the stimulus, the dot product of the stimulus vector u with the weight vector w.

So the firing rate of the output neuron is just w.u. So you can see that what this means is that the firing rate of the output neuron will be high if there is a high degree of overlap between the input, the pattern of the input, and the pattern of synaptic weights from the input layer to the output neuron.

We can see that w.u is big when w and u are parallel, are highly correlated, which means a neuron fires a lot when the stimulus matches the pattern of those synaptic weights.

Now, so you can see that for a given amount of power in the stimulus-- so the power is just the square magnitude of u-- the stimulus that has the best overlap with the receptive field, where cosine of that angle is 1, produces the largest response.

And so we now have actually a definition of the optimal stimulus of a neuron in terms of the pattern of synaptic weights. In other words, the optimal stimulus is one that's essentially proportional to the weight matrix. Any questions so far?

All right, so now let's turn to the question of how we use neural networks to do some interesting computation. So classification is a very important computation that neural networks do in the brain and actually in the application of neural networks for technology.

So what does classification mean? So how does the brain-- how does a neural circuit decide how a particular input-- let's say that it looks like you might eat it. How do we decide-- how do the neural circuits in our brain decide whether that thing that we're seeing is something edible or something that will make us sick based on past experience?

If we see something that looks like an animal or a dog, how do we know whether that's a friendly puppy or a or a wolf? So these are classification problems.

And feed forward circuits actually can be very good at classification. In fact, recent advances in training neural networks have actually resulted in feed forward neural networks that actually approach human performance in terms of their ability to make decisions like this.

All right. So basically, a feed forward circuit that does classification like this typically has an input layer. It has a bunch of inputs that represent sensory stimulus. And a bunch of output neurons that represent different categorizations of that input stimulus.

So you can have a retinal input here. Going to other layers of a network. And then at the end of that, you can have a network that starts firing when that input was a dog, or starts firing another neuron that starts firing when that input was a cat, or something else.

Now in general, classification networks that have one input layer and one output layer can't do this problem. You can't take a visual input and have connections to another layer of neurons that just light up when the picture that the network is seeing is a dog. Another neuron lights up when it's a cat.

Generally, there are many layers of neurons in between. But today, we're going to talk about a very simplified version of the classification problem and build up to the sorts of networks that can actually do those more complex problems.

So I just want to point out that the obviously our brains are very good at recognizing things. We do this all the time. There are hundreds of objects in every visual scene. And we're able to recognize every one of those objects.

But it turns out that there are individual neurons-- so in this case, I alluded to the idea that there are individual nones in this network that light up when the sensory input is a dog or light up when the input is an elephant. And it turns out that that's actually true in the brain.

So there have recently been studies where it's been possible to record in parts of the human brain in patients that are undergoing brain surgery for the treatment of epilepsy or tumors or things like that where you have to go in and find parts of the brain that are defective, find parts of the brain that are healthy. So when you do a surgery, you can be very careful to just do surgery on the damaged parts of the brain and not impact parts of the brain that are healthy.

So there are cases now, more and more commonly, where neuroscientists can work with neurosurgeons to actually record from neurons in the brain in these patients who are in preparation for surgery. And so it's been possible to record from neurons in the brain.

This was a study from Itzhak Frieds lab at UCLA. And this shows recording in the right anterior hippocampus. And what this lab did was to find neurons. So these were electrodes implanted in the brain. And then they basically take these patients and they show them thousands of pictures and look at how their brains respond to different visual inputs.

So let me just show you what you're looking at. These are just different pictures of celebrities. There's Luke Skywalker, Mother Teresa, and some others. This paper is getting old enough that you may not recognize most of these people.

But if you record from neurons in the brain, you can see that-- so what do you see here? I think that's Oprah. The image is flashed up on the screen for about a second.

You record this neuron spiking. Here you see a couple spikes. Here's when the image was actually presented. And here's where the image was turned off. You can see different trials.

So this neuron actually had a little bit of a response right there shortly after the stimulus was turned on. But you can see there's not that much response in these

neurons.

But when they flashed a different stimulus-- anybody know who that is? That's Halle Berry. Look at this neuron. Every time you show this picture, that neuron fires off a couple spikes very precisely.

If you look at the histogram, these are histograms underneath showing as a function of time relative to the onset of the stimulus, you could see that this neuron very reliably spikes. There's a different picture of Halle Berry. Neuron spikes. Different picture, neuron spikes. Another picture, neuron spikes.

A line drawing of Halle Berry, the neuron spikes. Catwoman, the neuron spikes. The text, Halle Berry, the neuron spikes. It's amazing.

So this group got a lot of press for this because they also found Jennifer Aniston neurons. They found other celebrities. This is like some celebrity part of the brain.

No, it's actually a part of the brain where you have neurons that have very sparse responses to a wide range of things. But they're extremely specific to particular people or categories or objects.

And it actually is consistent with this old notion of what's called the grandmother cell. So back before people were able to record in the human brain like this, there was speculation that there might be neurons in the brain that are so specific for particular things, that there might be one neuron in your brain that responds when you see your grandmother.

And so it turns out it's actually true. There are neurons in your brain that respond very specifically to particular concepts or people or things. So the question of how these kinds of neurons acquire their responses is really cool and interesting.

So that leads us to the idea of perceptrons. Perceptron is the simplest notion of how you can have a neuron that responds to a particular thing that detects a particular thing and responds when it sees it and doesn't respond when it doesn't.

So let's start with the simplest notion of a perceptron. So how do we make a neuron that fires when it sees something-- let's say a dog-- and doesn't fire when there is no dog?

So in order to think about this a little bit more, so we can begin thinking about this in the case where we have a single neuron input and a single output neuron. So if we have a single input neuron, then what comes in has to be-- it can't be an image right? An image is a high dimensional thing that has many thousands of pixels. So you can't write that down as a simple model with a single input neuron and a single output neuron.

So you need to do this classification problem in one-dimension. So we can imagine that we have an input neuron that comes from, let's say, some set of numbers-- I'll make up a story here-- some set of neurons that measure the dogginess of an input.

So let's say that we have a single input that fires like crazy when it sees this cute little guy here. And fires at a negative rate when it sees that thing, which doesn't look much like a dog. So we have a single input that's a measure of dogginess.

And now let's say that we take this dogginess detector and we point it around the world. And we walk around outside with our dogginess detector and we make a bunch of measurements. So we're going to see something that looks like this.

We're going to see a lot of measurements, a lot of observations down here that are close to zero dogginess. And we're going to see a bump of things up here that correspond to dogs. Whenever we point our dogginess detector at a dog, it's going to give us a measurement up here. And we're going to get a bunch of those.

And those things correspond to dogs. So we need to build a network that fires when the input is up here and doesn't fire when the input is down there. So how do we do that?

So the central feature of classification is this notion of binariness, of decision-making. That it fires when you see a dog and doesn't fire when you don't see a dog.

So there exists a classification boundary in this stimulus space. You can imagine that there's some points along this dimension above which you'll say that that input is a dog, below which you say that it isn't.

And we can imagine that that classification boundary is right here. It's a particular

number. It's a particular value of our dogginess detector, above which we're going to call it a dog, and below which we're going to call it something else.

How do we make this neuron respond by firing when there's a dog and not firing when there's no dog? Can we use a linear neuron? Can we use one of our linear neurons that we just talked about before?

We can't do that because a linear neuron will always fire more the bigger the input is. And it will fire less if the dogginess is 0. And it will even fire more negatively if the dogginess input is negative.

So a linear neuron is terrible for actually making any decisions. Linear neurons always go, ah, well, maybe that's a dog. Not really. There's no decisions.

So in order to have a decision, we need to have a particular kind of neuron. And that kind of neuron uses something very natural. In biophysics, it's the spike threshold of neurons. Neurons only fire when the input is above some threshold, generally. There are neurons that are tonically active. But let's not worry about those.

So many neurons only fire when the input is above some threshold. So for decision-making and classification, a commonly used kind of neuron takes this idea to an extreme. So for perceptrons, we're going to use a simplified model of a neuron that's particularly good at making decisions. There's no if, ands, or buts about it. It's either off or on. It's called a binary unit.

And a binary unit uses what's called a step function for its FI curve. That step function is 0-- the output is 0 if the input is zero or below. And the output is 1 if the input is above 0.

We can use that step function to create a neuron that responds when the input is above any threshold we want. So we can write down the output firing rate is this function, a step function-- that function of a quantity that's given by w times u, the synaptic weight times the input firing rate, minus that threshold.

So you can see if w times u, which is the input synaptic current, if that synaptic current is above theta, then this argument to this function is greater than 0, then the neuron spikes. If this argument is negative, then the neuron doesn't spike.

So by changing theta, we can put that decision boundary anywhere we want. Does that make sense?

Usually the way we do this is we pick a theta. We say our neuron has a theta of 1. And then we do everything else-- we do everything else we're going to do with this network with a theta.

So what I'm going to talk about today are just two cases. Where theta is a fixed number that's non-zero, or theta that's a fixed number that is equal to 0. So we're going to talk about those two cases.

So the neuron fires when the input w u is greater than theta. And it doesn't fire when it's less. So now the output neuron fires whenever the input neuron has a firing rate greater than this decision boundary.

So the decision boundary, the u threshold, is equal to theta divided by w. Does that make sense? U threshold is the neuron fires when u is greater than theta divided by w.

So the way we learn, the way this network learns to fire when that u is above this classification boundary is simply by changing the weight. Does that make sense?

So we're going to learn the weight such that this network fires whenever the input says there's a dog. And it doesn't fire whenever the input says there's no dog.

So let's see what happens when w is really small. If w is really small, then what happens is all of these-- remember, this is the input. That's that the dogginess detector.

If w is really small, then all these inputs get collapsed to a small input current into our output neuron. Does that make sense? So all those different inputs, dogs and non-dogs, gets multiplied by a small number. And all those inputs are close to 0.

And if all those inputs are close to 0, they're all below the threshold for making this neuron spike. So this network is not good for detecting dogs because it says it never fires, whether the input is a dog or a non-dog.

Now what happens if w is too big? If w is really big, then this range of dogginess

values gets multiplied by a big number. And you can see that a bunch of non-dogs make the neuron fire. Does that make sense?

So now this one fires for dogs plus doggie-ish looking things, which, I don't know, maybe it'll fire when it sees a cat. That's terrible.

So you have to choose w to make this classification network function properly. Does that make sense? And if you choose w just right, then that classification boundary lands right on the threshold of the neuron. And now the neuron spikes whenever there is a dog. And it doesn't spike whenever there's not a dog.

So what's the message here? The message is we can have a neuron that has this binary threshold. And what we can do is simply by changing the weight, we can make that threshold land anywhere on this space of inputs.

And we can actually use the error to set the weight. So let's say that we made errors here. We classify dogs as non-dogs because the neuron didn't fire. You can see that this was the case when w was too small.

So if you classify dogs as non-dogs, then you need to make w bigger. And if you classify non-dogs as dogs, you need to make w smaller. And by measuring what kind of errors you make, you can actually fix the weights to get to the right answer.

So this is a method called supervised learning where you set w randomly. You take a guess. And then you look at the mistakes you make. And you use those mistakes to fix the w.

In other words, you just look at the world and you say, oh, that's a dog. And then your mom says, no, that's not a dog, that's something else. And you adjust your weights.

I think that was the example I just gave. You're going to make that w smaller. In another case, you'll make the other kind of mistake and you'll fix the weights.

So this is called a perceptron. And the way you learn the weights in a perceptron is you just classify things and you figure out what kind of mistake you made and you use that to adjust the weights. So that's the basic idea of a perceptron and perceptron learning. And there's a lot of mathematical formalism that goes into

how that learning happens. And we're going to get to that in more detail in the next lecture.

But before we do that, I want to go from having a one-dimensional case. So here we had a one-dimensional network that was just operating on dogginess. And then we have a single neuron that says, was that a dog or not.

But in general, you're not classifying things based on one input. Like for example when you have to identify a dog, you have a whole image of something. And you have to classify that based on an image.

So let's go from the one-dimensional case to a two-dimensional case. So the classification isn't done on one-dimension, but it's based on many different features.

So let's say that we have two features, furriness and bad breath. That dog doesn't really look like it has bad breath. but mine does. So you can have two different features, furriness and bad breath. And dogs are generally, let's say, up here.

Now you can have other animals. This guy is definitely not furry. So he's down here somewhere. And you can have this guy up here. He's definitely furry.

So you have these two dimensions and a bunch of observations in those two dimensions, in those higher dimensions. And you can see that, in this case, you can't actually apply that one-dimensional decision-making circuit to discriminate dogs from these other animals.

Why is that? Because if I apply my one-dimensional perceptron to this problem, you can see that I could put a boundary here and it will misclassify some of these non-furry animals as dogs. Or I could put my classifier here and it will misclassify some of these cats as dogs.

So how would I separate dogs from these other animals if I had this two-dimensional space? What would I do? How would I put a classification bound? If this doesn't work and this doesn't work, what would I do?

You could put a boundary right there. So in this little toy problem, that would perfectly separate dogs from all these non-dogs. So how do we do that?

Well, what we want is some way of projecting these inputs onto some other direction so that we can put a classification boundary right there. And it turns out there's a very simple network that does that. It looks like this.

We take each one of those detectors, a furriness detector and a bad breath detector, and we have those two inputs. We have those inputs synapse onto our output neuron with some weight w1 and some weight w2, and we calculate the firing rate of this neuron.

Now we have this problem of how do we place this decision boundary correctly. What's the answer? Well, in the one-dimensional example, what is it that we learned? What was it that we were actually changing?

We were taking guesses. And if we were right or wrong, we did what? We changed the weight. And that's exactly what we do here.

We're going to learn to change these weights to put that boundary in the right place. If we just take a random guess for these weights, that line is just going to be some random position. But we can learn to place that line exactly in the right place to separate dogs from non-dogs.

So let's just think a little bit more about how that decision boundary looks as a function of the weight. So let's look at this case where we have two inputs. So now you can see that the input to this neuron is w.u.

So now if we use our binary neuron with a threshold, we can see that the firing rate of this output neuron is this step function operating on or acting on this input, w.u minus theta. So now what does that look like? The decision boundary happens when this quantity is pulled to 0. When this input is greater than 0, the neuron fires. When this input is less than 0, it doesn't fire.

So what does that look like? So you can see the decision boundary is when w.u minus theta equals 0. Does anyone know what that is?

Remember, u is our input space. That's what we're asking, where is this decision boundary in the input space. w is some weights that are fixed right now, but we're gradually going to change them later.

So what is that an equation for? It's a line. That's an equation for a line.

If u is our input, you can see w.u equals theta. That's an equation for a line, base of u. The slope and position of that line are controlled by the weights w and the threshold theta.

So you can see this is w1, u1, plus w2, u2 equals theta. In the space of u1 and u2, that's just a line.

So let's look at the case where theta equals 0. You can see that if you have this input space, u1 and u2, if you take a particular input u and dot it into w-- so let's just pick a w in some random direction-- the neuron fires when the projection of u along w is positive. So you can see here, the projection of u along w is positive.

So in this case for this u the neuron will fire. So any u that has a positive projection along w will make the neuron spike. So you can see that all of these inputs will make the neuron spike.

All of these inputs will make the neuron not spike. Does that make sense? So you can see that the decision boundary, this boundary between the inputs that make the neuron spike and the inputs that don't make the neuron spike, is a line that's orthogonal to w. Does that make sense?

Because you can see that any u, any input, along this line will have zero projection, will be orthogonal to w. Will have zero projection. And that's going to correspond to that decision boundary.

So let's just look at a couple of cases. So here a set of points that correspond to our non-dogs. Here are a set of points that correspond to our dog. You can see that if you have a w in this direction, that produces a decision boundary that nicely separates the dogs from the non-dogs.

So what is that w? that w is 1, comma, 0. And we're going to consider the case where theta is 0.

Let's look at this case here. So you can see that here are all the dogs. Here are all the non-dogs. You can see that if you drew a line in this direction, that would be a

good decision boundary for that classification problem. You can see that a w corresponding to solving that problem is 1, comma, minus 1, and theta equals 0.

Let's look at the case where theta is not 0. So here we have w.u minus theta. When theta is not 0, then the decision boundary is w.u equals some non-zero theta.

That's also a line. It's a equation for a line. When theta is 0, that decision boundary goes through the origin. When theta is not 0, the decision boundary is offset from the origin.

So we could see that when we had theta is 0, the decision boundary-- that network only works if the decision boundary is going through the origin. In general, though, we can put the decision boundary anywhere we want by having this non-zero theta.

So here's an example. Here are a set of points that are the dogs. Here are a set of points that are the non-dogs. If we wanted to design a network that separates the dogs from the non-dogs, we could just draw a line that cleanly separates the green from the red dots.

And now we can calculate w that gives us that decision boundary. How do we do that? So the decision boundary is w minus u.theta.

Let's say that we want to calculate this weight vector w1 and w2. And let's just say that our neuron has a threshold of 1. So we can see that we have two points on the decision boundary.

We have one point here, a, comma, 0, right there. We have another point here, 0, comma, b. And we can calculate the decision boundary using ua.w equals theta, ub.w equals theta. That's two equations and two unknowns, w1 and w2.

So if I gave you a set of points and I said calculate a weight for this perceptron that will separate one set of points from another set of points, and I give you a theta for the output neuron, all you have to do is draw a line that separates them, and then solve those two equations to get w1 and w2 for that network.

It's very easy to do this in two dimensions. You can just draw a line and calculate the w that corresponds to that decision boundary. Any questions about that? Just that, if you have questions, you should ask because that's going to be a question you ought

to solve.

So you can see in two dimensions you can just look at the data, decide where's the decision boundary, draw a line, and calculate the weights w.

But in higher dimensions, it's a really hard problem. In high dimensions, first of all, remember in high dimensions you've got images. Each pixel in that image is a different dimension in the classification problem.

So how do you write down a set of weights? So imagine that's an image, that's an image. And you want to find a set of weights so that this neuron fires when you have the dog, but doesn't fire when you have the cat.

That's a really hard problem. You can't look at those things and decide what that w should be. So there's a way of taking inputs and taking the answer, like a 1 for a dog and a 0 for non-dogs, and actually finding a set of weights that will properly classify those inputs.

And that's called the perceptron learning rule. And we're going to talk about that in the next lecture.

So that's what we did today. And we're going to continue working on developing methods for understanding neural networks next time.