HAIM SOMPOLINSKY: My topic today is discussing sensory representations in deep cortex-like architectures. I should say the topic is perhaps toward a theory of sensory representations in deep networks. As you will see, our attempt is to develop a systematic theoretical understanding of the capacity and limitations of architectures of that type.

The general context is well known. In many sensory systems, we see information is propagating from the periphery, like the retina, to primary visual cortex, and then, of course, many stages up to a very high level, or maybe the hippocampal structure. It's not purely feedforward. There are backward, massive backward or top-down connections, the recurrent connections, and some of those extra features I'll talk about.

But the most intuitive feature of that is simply a transformation or filtering of data across multiple stages. Similarly in auditory pathway. In other systems, we see a similar structure or aspect of similar structure as well. A well-known and a classical formative system for computational science is cerebellum, where you have information coming from the mossy fiber layer, then expand enormously into a granule layer, and then converge to a Purkinje cell.

So if you look at a single Purkinje cell, the output of the cerebellum, as a unit, then you see there is, first, an expansion from 1,000 to two orders of magnitude higher in the granule layer. And then convergence of 200,000 or so parallel fibers onto a single Purkinje cell. And there are, those type of modules, many, many across the cerebellum. So, again, a transformation which involves, in this case, expansion and then convergence.

In the basal ganglia, which I wouldn't categorize it as a sensory system. More related to motor. Nevertheless, you see cortex converging first to various stages of the basal ganglia, and then expanding again to cortex.

Hippocampus has also multiple pathways, but some of them include a convergence. For instance, convergence to a CA3, and then expansion again to cortex. But there are other multiple pathways as well, different stages of sensory information propagating, of course, across them.

And, finally, the artificial network story of deep neural networks, all of you may have heard. Input layer, then sequence of stages. Purely feedforward. And at least the canonical leading network is one that the output layer has object recognition, object classification task. And the whole network is studied by backprop, supervised learning for that.

What I'll talk about is more in the spirit of the idea that the first stages are more general purpose than the specific classic task in the output layer. So there are many issues. Number of stages that are required, the size of them, why compression or expansion.

In many systems, you'll see that the fraction of active neurons is small in the expanded layer. That's what we call sparseness. So high sparseness means small number of neurons active at any given stimulus. It's just the terminology is somewhat confusing. So high sparseness is small number of active neurons.

One important and crucial question is how to transform. What are the filters, the weights that are good for transforming sensory information from one layer to another? And, in particular, whether random weights is good enough, or maybe even optimal in some sense. Or one needs more structure, the more learned type of synaptic way.

This is a crucial question, perhaps not for machine learning but for computational neuroscience, because there is some experimental evidence, for at least of some of the systems that are studied, that the mapping from the compressed, original representation to the sparse representation is actually done by randomly connected weights.

So one example is olfactory cortex. The mapping of olfactory representation from the olfactory bulb, so from glomerulus layer, to the piriform cortex seems to be random, as far as one can say.

Similarly, in the cerebellum, the example that I mentioned before, when one looks at the mapping from the mossy fiber to the granule cell, again enormous expansion by a few orders of magnitude. Nevertheless, they seem to be random.

Now, of course, one cannot say exclusively that they are random and there are no subtle correlations or structures. But, nevertheless, there is a strong motivation to ask whether random projections are good enough. And if not, what does it mean structured? What kind of structure is appropriate for this task?

Question of top-down and feedback loops, recurrent connections, and so on. So that's all I hope to, at least briefly, mention later on in my talk.

So before I continue, most of, or a large part of the talk will be based on published and unpublished work with Baktash Babadi, who was, until recently, a postdoctoral, a Swartz Fellow at Harvard University. Went to practice medicine. Elia Frankin, a master student at the Hebrew University. SueYeon, who all you know here, at Harvard. Uri Cohen, a PhD student at the Hebrew University. And Dan Lee from Penn University.

So here is our formalization of the problem. We have an input layer, denoted 0. Typically, it's a small, it's a compressed layer with dense representation. So here, every input will generate maybe half of the population, on average,

Then there is a feedforward layer of synaptic weights, which expand to a higher-dimension layer, which we call cortical layer. It's expanded in terms of the number of neurons. So this will be S1. It is sparse because the f, the fraction of neurons that are active for each given input vector, will be small. So it is expanded and sparse.

That will be the first part of my talk, discussing this. Then, later on, I'll talk about staging, cascading this transformation to several stages. And ultimately there is a readout, will be some classification task. 1 will be one classification. 2 will be another classification rule, et cetera, each one of them with synaptic weights which are learned to perform that task. So we call that a supervised layer, and that's the unsupervised layer.

So that's a formalization of the problem. And, as you will see, we'll make enormously simplifying abstraction of the real biological system in order to try to gain some insight about the computational capacity of such systems.

So the first important question is what is the statistics, the statistical structure of the input? So the input is kind of n-dimensional vector, where n, or n0, when n is the number of units here. So each sensory event evokes a pattern of activity here.

But what is the structure, the statistical structure, that we are working with? And the simplest one that we are going to discuss is the following. So we assume, basically, that the inputs are coming from a mixture of, so to speak, a mixture of Gaussian statistics. It's not going to be Gaussian because, for simplicity, we'll assume they are binary. But this doesn't matter actually.

So imagine that this is kind of a graphical represent-- a caricature of high-dimensional space. And imagine the inputs, the sensory inputs, imagine that they are clustered around templates or cluster centers. So these will be the centers of these balls. And the input itself is coming from the neighborhoods of those templates. So each input will be one point in this space, and it will be originating from one of those ensembles of one of those states.

So that's a simple architecture. And we are going-- in real space, it will be mapping from one of those states into another state in the next layer. And then, finally, the task will be to take-- imagine that some of those balls are classified as plus. Let's say the olfactory factory stimuli, and some of them are classified as appetitive, some of them as aversive.

So the output layer, the readout unit, has to classify some of those spheres as a plus, and some of them are minus. And, of course, depending on how many of them are in a dimensionality, and their location, this may or may not be an easy problem.

So, for instance, here it's fine. There is-- a linear classifier on the input space can do it. Here, I think there are, there should be some mistakes here. Yeah, here.

So here is a case where the linear classifier at the input layer cannot do it. And that's our-- that's a theme which is very popular, both in computation neural science and system neural science studies in machine learning. And the following question comes up.

Suppose we see that there is a transformation of data from, let's say, a photoreceptor layer in vision to the ganglion cells at the output of the retina, then to cortex in several stages. How do we gauge, how do we assess what is the advantage for the brain to transform information from one, let's say, from retina to V1, and so on and so forth.

After all, in this feedforward's architecture, no net information is generated at the next layer. So if no net information is generated, the question is, what did we gain by these transformations? And one possible answer is that it is reformatted, reformatting the sensory representation into different representation which will make subsequent computations simpler. So what does it mean, subsequent computation is simpler?

One notion of simplicity is whether subsequent computation can be realized by a simple linear readout. So that's the strategy that we are going to adopt here. And this is to ask, as the information, as the representation is changing as you go from one layer to another, how well a linear readout will be able to perform the task.

So that's the input. That's the story. And then, as I said, there is an input, unsupervised representations, and supervised at the end.

I need to introduce notations. Bear with me. This is a computational talk. I cannot just talk about ideas, because the whole thing is to be able to actually come up with a quantitative theory that tests ideas. So let me introduce notations.

So at the centers, at each layer, you can ask what is the representation of the centers of these stimuli? And I'll denote the center by a bar. And mu is index of the patterns. So mu goes from 1 to P. P Is the number of those balls, number of those spheres, or number of those clusters, if you think about clustering some sensory data. So P would be the number of clusters.

i, from 1 to N, is simply the neuron or the unit activation at each mu. And L is the layer. So 0 is the input layer. It's up to L layer.

So this would be 0, 1. The mean activation at each layer from 1 on will just have to be a constant to be f. f goes from 0 to 1. The smaller f is, the sparser the representation is. We will assume that the input representation is dense. So this is 0.5.

N, again, we'll assume to be, for simplicity, a constant across layers, except for the first layer, where there is expansion. You can vary those parameters, and actually the theory accommodates variations of those. But that's the simplest architecture. You expend a dense representation into a sparse higher dimension, and you keep doing it as you go along. So that's notation.

Now, how do we assess what is the next stages doing to those clusters. So, as I said, one measure is take a linear classifier and see how linear classifier performs. But, actually, you can also look at the statistics of the injected sensory stimuli at each layer and learn something from it.

And, basically, I'm going to suggest looking at two major statistical aspects of the data in each layer of the transformation. One of them is noise, and one of them is correlation. So what is noise? So, again, noise will be simply the radius, or measure of the radius of the sphere.

So if you had only the templates as inputs, the problem would be simple. Problem would be easy as long as we have enough dimension. You expand it. You can easily do linear classifier and solve the problem. So the problem, in our case, is the fact that the input is actually the infinite number of inputs, or exponentially large number of possible inputs, because they all come from a Gaussian or a binarized version of a Gaussian noise around the templates.

And I'll denote the noise by delta. 0 means no noise. The normalization is such that 1 means that they are random. So delta equals to 1 means that, basically, you cannot tell, the input, whether it's coming from here or from any other points in the input space.

The other thing, correlations, is more subtle. So I'm going to assume that those balls are coming from kind of uniform distribution. Imagine you take a template here. You draw a ball around it. You take a template. Here, you draw a ball. Everything is kind of uniformly distributed.

The only structure is the fact that data comes from this mixture of Gaussians or noisy patterns around those centers. So that's fine. But as you project those clusters into the next stage, I claim that those centers, those templates, get new representation, which can actually have structure in them, simply by the fact that you put all of them into this common synaptic weights into the next layer.

And I'm going to measure this by Q. And, basically, low Q or 0 Q is basically a kind of randomly uniformly distributed centers. And I'll always start from that at the input layer. But then there is a danger, or it might happen that, as you propagate this information or this representation through the next layer, the centers will look like that, or the data, structure of the data, looks like that.

So, on average, the distance between two centers, on average, is the same as here. But they are clumped together. It's kind of random clustering of the clusters. And that can be induced by the fact that the data is feedforwarded from this representation.

That can pose a problem. If there is no noise, then there is, again, no problem. You can differentiate between them, and so on. But if there is noise, this can aggravate the situation, because some of the clusters become dangerously close to each other. And we will come to it.

But, anyway, so we have this delta, the noise, the size of the clusters, and we have Q, the correlations, how they are clumped in each representation. And now we can ask how delta evolve when you go from one presentation to another, how Q evolve from one presentation to another, and how linear classifier performance will change from one representation to another.

So the simplicity of this assumption allows for a kind of systematic, analytical exploration or study of this. These are definitions. Let's go on.

So what will be the ideal situation? So the ideal situation will be that I start from some level of noise, which is my spheres at the input layer. I may or may not start with some correlation. The simplest case would be that I start from randomly distributed centers. So this would be 0.

And the best situation will be that, as I propagate the sensory stimuli, delta, the noise, will go to 0. As I said, if the noise goes to 0, you are left with basically points. And those points, if there is enough dimensionality, those points would be easily classifiable.

It would also be good, if the noise doesn't go to 0, to have also kind of uniformly spread clusters. So it will be good to keep Q to be small.

So let's look at one layer. So let's look at this. We have the input layer, the output layer here, and the readout. The first question is what to choose for this input layer. So the simplest answer would be choose random.

So what we do, we just take Gaussian. The Gaussian weights in this layer are very simple. 0 mean, with some normalization. It doesn't matter.

Then we project them into each one of these guys here. And then we add threshold to enforce the sparsity that we want. So whatever the activation here is, whatever the input here is, the threshold makes sure that only the f with the largest input will be active, and the rest will be 0.

So there is a nonlinearity, which is of course extremely important. If you map one layer to another with a linear transformation, you don't gain anything in terms of classification. So there is a nonlinearity, simply a threshold nonlinearity after an input projection.

All right. So how we are going to do this? So it's straightforward to actually compute analytically what will happen to a noise. So imagine you take two input vectors with some Hamming distance apart from each other. You map them by convolving them, so to speak, with Gaussian weights, and then thresholding them to get some sparsity.

So f is the sparsity. The smaller the f is, the sparser it is. So this is the noise level, the normalized Gaussian-- I'm sorry-- sphere radius or Hamming distance in the output layer, and also the input layer. Well if 0 at start, then of course you start at the origin. If you are random at the input, you will be random there. So these points are fine.

But, as you see, immediately there is an amplification of the noise as you go from the input to the output. So you start from 0.2, but you get actually, after one layer, to 0.6. And, actually, the sparser it is-- so this is a relatively high sparsity, or at least you go from here to here by increasing sparsity, namely f becomes smaller.

And as f becomes smaller, this curve is actually steeper and steeper. So not only you amplify noise, but you also-- the amplification becomes worse the sparser the representation is. So that is the kind of negative result.

The idea that you can gain by expanding data to a higher dimension and make them more separable later on dates back to David Marr's classical theory of the cerebellum. But what we show here is that, if you think not about clean data, a set of points that you want to separate, but you think about the more realistic case of you have noisy data, or data with high variance, then the situation is very different. So a random expansion actually amplifies those.

And that's a theme that will-- actually, we will live with it as we go along. Random expansion is doing the separation of the templates. But the problem is it also separates two nearby points within a cluster. It also separates them. So everything becomes separated from each other. And this is why noise is amplified.

Now, what about the most subtle thing, which is the kind of overlap between the centers? So, on average, the centers are as far apart as random things. But if you look, not on average, but you look at the individual pairs, you see that there is an excess correlations or overlap between them.

So this is overlap between the centers. Again, on average, it is 0, but the variance is not 0. On average it's like random, but the variance is different, is larger than random. And there is an amplification. There is a generation of this excess overlap, although it's nicely controlled by sparsity.

So as sparsity goes down, these correlations go down. So that's not a tremendous problem. The major problem, as I said, is the noise.

By the way, you can nicely do an exercise where you generate, you look at this cortical layer representation, and you do SVM, or PCA, and you look at the eigenvalue spectrum. So if you just look at random sparse points, and you look at the SVD this is the eigenvalues number ranked-- then that's what you find. It's the famous Marchenko-Pastur distribution.

But, in our case, you see there is an extra power. In this case, the input layer is 100, so the extra power in the input layer, in the first input eigenvalue. Now, why is it so?

What Q is telling us, what nonzero Q is telling us is the following. You take a set of random points, and you project them into higher dimensions. You start with 100 dimensions, and you project them in 1,000 dimensions. On average, they are random.

But actually-- so you would imagine that it's a perfect thing. You project them with random weights. Then you would imagine that you just created a set of random points in the expanded dimension representation. If this was so, then if you do SVM or PCA on this representation, you will find what you expect from a PCA of a set of random points. And this is this one.

In fact, there is a trace of low dimensionality in the data. So I think that's an important point, which I would like to explain. You start from a set of points. If you don't threshold them and you just map them into 1,000-dimensional space, those 100-dimensional input will remain 100-dimensional. Just be rotated, and so on, but everything will live in 100-dimensional space.

Now you add thresholding, high thresholding by sparsity. So those 100-dimensional subspace becomes now 1,000-dimensional sparsity because of the nonlinearity. But this nonlinearity, although it takes 100-dimensional input and makes them 1,000-dimensional, it's still not like random.

This 1,000-dimensional cloud is still elongated. It's not simply uniformly distributed. And this is the signature that you see here. In the first largest 100 eigenvalues, there is extra power relative to the random.

The rest is not 0. So if you look here, this goes up to 1,000. The rest is not 0. So the system is, strictly speaking, 1,000-dimensional space, but it's not random. It has increased power in 100 channels.

If you do a readout, a linear classifier readout, what you find in this-- again, when you expand with random weights, you find that there is an optimal sparsity. So this is the readout error. For a classifier, the output is a function of the sparsity for different levels of noise.

And you see that, in the case of random weights, there is a very high sparsity, is bad. There Is an optimal sparsity or sparseness, and then there is a shallow increase in the error when you go to a denser representation.

One important point which I want to emphasize coming from the analysis-- let me skip equations-- and this is what you see here. The question is, can I do better by further increase the layer? So here I plot the readout error as a function of the size of cortical layer. Can I do better? If I make the kernel dimensionality infinite, can I do better?

Well, it can do better if you start with 0 noise. But if you have noisy inputs, then basically, the performance saturates. And that's kind of surprising. We were expecting that, if you go to a larger and larger representation, eventually the error will go to 0. But it doesn't go to 0. And that actually happens even for what we call structured representation.

And that's the same for different types of readout-- perceptual, and pseudo-inverse, SVM. All of them show this saturation as you increase the size of cortical layer. And that's one of the very important outcome of our study. That when you talk about noisy inputs, you can think about it as kind of more generalization task. Then there is a limit about what you gain by expanding representation.

Even if you expand in a nonlinear fashion and you increase the dimensionality, you cannot combat the noise, at least up to some level. Beyond some level, there is no point of further expansion, because basically the error saturates

Let me, since time goes fast, let me talk about the alternatives. So if random weights are not doing so well, what are the alternatives? The alternative is to do some kind of unsupervised learning. Here we are doing it in a kind of a shortcut of unsupervised learning.

What is the shortcut? We say the following. Imagine that these layers, the learner knows about the representation of the clusters. It doesn't know the labels. In other words, whether those are pluses and those are minuses, which one are pluses and minuses. But he does know about the statistical structure of the input, and this is this S bar. These are the centers.

So we want to encode the statistical structure of these input in this expansion of the weights. And the way we do the simplest way is the kind of Hebb rule. We do the following. We say let's first choose, or recruit, or allocate a state, a sparse state here, randomly chosen, to associate, to represent each one of the clusters. So these are the R. R are the randomly chosen patterns here.

And then we associate between those randomly chosen representations and the actual centers of the clusters of the inputs. So this is S bar and R. And then we do the association by the simple, what's called Hebb rule.

So this Hebbian rule associates cluster center with a randomly assigned state in the cortical layer in a kind of simple summation or outer product for the Hebb rule. There are more sophisticated ways to do it, but that's the simplest one of doing it.

So it turns out that this simple rule has enormous potential for suppressing noise. So, again, this is the input noise and the output noise. The Hamming distance of the input and the output properly normalized. And you see that, as you go to higher and higher sparseness, to lower and lower f, this is basically the input noise is completely quenched when f is large.

When f is 0.01, for instance, this is this already. Sub-linear when f 0.05 is here, and so and so forth. So sparse representation, in particular, are very effective in suppressing noise, but provided the inputs have kind of unsupervised learning encoded into them which embed into them the cluster structure of the inputs.

The same or similar thing is true for Q for these correlations. If you look at the-- this was a random correlation. This is a function of f, and this is Q, the correlation. It's extremely suppressed for sparse representation. Basically, it's exponentially small with 1/f, so it's basically 0 for sparse representation.

Which means that those centers look like randomly distributed, essentially, and with very small noise. So you took these spheres and you basically map them into random points with a very small radius. So it's not surprising that, in this case, the error for small f-- the error, even for large noise values, the error is basically small, 0.

Nevertheless, it is still saturating as a function of the network size, of the cortical size. So the saturation of performance as a function of cortical size is a general property of such systems. Nevertheless, the performance itself for any given size is extremely impressive, I would say, when the system is sparse and the noise level is kind of moderate.

OK, let me skip this because I don't have time. Let me briefly talk about extension of this story to multi-layer. So we are now briefly discussing what happens if you take this story and you just propagate it as you go along the architecture.

So let's start with random weights. So the idea is maybe something is good happening.

Although initially performance was poor, maybe we can improve the performance by cascading such layers. And the answer is no, particularly the noise level.

This is now the number of layers. What we discussed before is here. And you see the problem becomes worse and worse. As you continue to propagate those signals, the noise is amplified and essentially goes to 1. So basically you will get just random performance if you keep doing it with random weights.

The reason-- where is it? I missed a slide. The reason is, basically, that if you think about the mapping from one layer of noise to another layer of noise, there are two fixed points, 0 and 1. The 0 fixed point is unstable. Everything goes eventually to 1.

So it is a nice-- this system gives you a nice perspective about this deep network by thinking about it as a kind of dynamical system. For instance, what is the level of noise at one layer, how it's related to the level of noise at previous layer. So it's kind of iterative map. Delta n versus delta n minus 1.

And what's good about it is, once you kind of draw this curve, one layer is mapped to another layer, you can know what happens to a deep network. We could just iterate this. You have to find what are the fixed points, and which one is stable and which one is not. In this case, the 1 is stable, the 0 is unstable. So, unfortunately, from any level of noise that you will start, you eventually go to 1.

Correlations, is a similar story, but-- and the error will go to 0.5. So that's very well. There are cases, by the way, that you can find parameters where initially you improve, like here. But then eventually it will go to 0.5.

Now, if we do similar-- if we compare this to what happened to the structured weights if you keep doing the same kind of unsupervised Hebbian learning from one layer to another-- and I'll skip the details-- you see the opposite. So here are parameter value in which one stage of the expansion stage is actually increasing the noise.

And this is because f is not too small, and the load is large, and the noise is starting. So you can have such situation. But even in such situation, eventually the system goes into stages where the noise basically goes to 0.

And if you compare the story why it is so to kind of iterative map picture, you see that the picture is very different. You have one fixed point at 0. You have one fixed point at 1. You have intermediate fixed point at high value. But this is an unstable fixed point, and both of them are stable fixed points.

So if you start from even from large values of noise, eventually you will iterate to 0. So it does buy you to actually go into several stages of this deep network to make sure that the noise is suppressed to 0.

Similarly for the correlations. Even if the parameters are such that initially correlations are increased, and you can find parameters like that, eventually correlations will go to almost 0.

And this is comparison of the readout error as a function of the layers with structured weights, and I compare it with the readout error of infinitely wide layer, kind of a kernel with infinitely wide kernel. And you can see that, at least for-- here I compare the same type of unsupervised learning but two different architectures. One is deep network architecture, and the another one is shallow architecture, infinitely wide.

I'm not claiming that we can show that there is no kernel or shallow architecture which will do better, but I'm saying if we compare the same learning rule but with the two different architectures, you'll find that you do gain by going into multiple stages of nonlinearity than by using an infinitely wide layer.

I'll skip this. I want to go briefly to two more issues. One issue is the recurrent networks. Why recurrent networks? The primary reason is that, in each one of those stages that I refer to, if you look at the biology, on most of them-- not all of them but most of them, and definitely in neocortex-- you find massive recurrent or lateral interactions between each one of the layers.

So, again, we would like to ask, what is the computational advantage of having this recurrent layer. Now, in our case, we had an extra motivation, and this is-- remember that I started in saying that, in some cases, there is experimental evidence that the initial projection is random. So that we ask ourselves, what happens if we do this.

If we start from random projection, feedforward projection, and then add recurrent connections. Think about it as from the olfactory bulb, for instance, to piriform cortex, perhaps random feedforward projections. But then the association, recurrent connections in piriform cortex are structured.

How do we do that? We start, we imagine starting from random projection, generating initial representation by the random projection, and then stabilizing those representation into attractors by the recurrent connections. And that actually works pretty well. It's not the optimal architecture, but it's pretty well. For instance, noise, which is initially increased by the random projections, were quenched by convergence to attractors.

And, similarly, Q will not go to 0, but will not continue growing, but will go to an intermediate layer. And the error is pretty well. So if you look at in this case, the error really goes down to very low values. But now it's not layers. Now it is the number of iterations of the recurrent connections. So you start from just input layer, or random projection, and then you iterate the dynamics and it goes to 0. So it's not the layers. It's just the dynamics of the convergence to attractor.

My final point. I have 3 or 4 minutes? OK. My final point before wrapping up is the question of top-down. So recurrent, we briefly talked about it. But incorporating contextual knowledge is a major question. How can you improve on deep networks by incorporating, not simply the feedforward sensory input, but other sources of knowledge about this particular stimulus?

And it's important that we are not talking about knowledge about the statistics of the input which can be incorporated into the learning of the feedforward one. But we're talking about inputs which are, or knowledge, which we have now on the network which already has learned whatever it has learned.

So we have a mature network, whatever the architecture is. We have a sensory input. It goes feedforward. And now we have additional information, about context for instance, that we want to incorporate with the sensory input to improve the performance. So how do we do that?

It turns out to be non-trivial computational problem. It is very straightforward to do it in Bayesian framework, where you simply update the prior of what the sensory input is by this contextual information. But if you want to implement it in the network, you find that it's not easy to find the appropriate architecture.

So I'll just briefly talk about how we do it. So imagine you have, again, these sensory inputs, but now there is some context, different contexts. And imagine you have an information that the input is coming from that particular part of state space.

So basically the question is how to amplify selectively a specific set of states in a distributed representation. So usually when we talk about attention, or gating, or questions like that, we think about, OK, we have these neurons. We suppress those, or maybe amplify other ones. Or we have a set of axons, or pathways. We suppress those, and amplify those.

But what about a representation which is more distributed where you have to really suppress states rather than neural populations. So I just won't go-- again, it's a complicated architecture. But, basically, we're using some sort of a mixed representation, where we take the sensory input and the category or contextual input, mix the nonlinearity, use them to clean it, and propagate this.

So it's a more complicated architecture, but it works beautifully. Let me show you here an example, and you'll have a flavor of what we are doing. So now the input, we have those 900 spheres or templates, but they are organized into 30 categories, and 30 tokens per category.

Now, the tokens, which are the actual sensory inputs, are represented by, let's say, 200 neurons. And you have a small number of neurons representing a category. Maybe 20 is enough. So that's important, and you don't have to really expand dramatically the representation.

So this is the input. And now we have very noisy inputs. If you look at the readout, this is layers here, and there is readout error. If you do it on the input layer, or any subsequent layer here, but without top-down information. With structured interactions and all that I told you, this is such a noisy input where the performance is basically 0.5. There is nothing that you can do without top-down information in this network.

You can ask what will be the performance. If you have an ideal observer that looks at the noisy input and makes maximum likelihood categorization. Well, then it will do much better. Also not 0, so this is at this level. This higher error is in virtue of the fact that this network is still not doing what an optimal maximum likelihood observer will do.

So this is the network. This is a maximum likelihood readout, both of them without extra top-down information. And in the network that I kind of hinted about, if you add this top-down information by generating mixed representation, you get a performance which is really dramatically improved. And as you keep doing it one layer from another, you really get a very nice performance.

So let me just summarize. There is one more before summarizing. Yeah, OK. Before that. OK.

So two points to bear in mind. One of them is that what I discussed to you today relies on assuming either random, comparing random projection, to unsupervised learning of a very simple type, of a kind of Hebbian type. The output can be Hebbian, or perceptron, or SVM, and so on.

You could ask, what happens if you use learning rules, more sophisticated learning rules for the unsupervised weights? Some of them we've studied. But, anyway, that's something which is important to explore. And another very important issue for thinking about object recognition in vision and in other real-life problem is input statistics.

Because what we assumed is a very simple mixture of Gaussian model. So you can think about the task of the network is to take the invariance, which is the variation away from the center of the spherical variation, and to generate representation which is invariant to that. But this is a very simple invariance problem, because the invariance was simply restricted to these simple geometric structures.

More problems which are closer to what real-life problems are will have inputs which are, essentially, have some structure, but the structure can be of a variety of shapes. Each one of them correspond to an object, or a cluster, or a manifold representing an entity, a perceptual entity.

But how you go from this nice, simple problem of this spherical invariance problem to those problems, it's of course a challenging problem. And that's the work which we are now, ongoing work, also with SueYeon Chung and Dan Lee. But it's a story which is still at the stage of unfolding.