Honda Foundation Report No. 179
Commemorative lecture at the 40th Honda Prize
Award Ceremony on the 18th November 2019

The Deep Learning Revolution

Dr. Geoffrey E. Hinton

Professor Emeritus, the University of Toronto

Chief Scientific Advisor, Vector Institute

Engineering Fellow, Google Research

HONDA FOUNDATION

Dr. Geoffrey E. Hinton

Professor Emeritus, the University of Toronto
Chief Scientific Advisor, Vector Institute
Engineering Fellow, Google Research

■Date of Birth

December 6th, 1947

■Education and Training

1970: Cambridge University, BA in Experimental Psychology

1978: University of Edinburgh, Ph.D. in Artificial Intelligence

■Employment History

1978–80: University of California, San Diego, Postdoctoral Fellow
1980–82: MRC Applied Psychology Unit, Cambridge, Research Scientist
1982–87: Carnegie Mellon University, Assistant & Associate Professor
1987–current: University of Toronto, Professor, Computer Science Department
1998–2001: University College London, Director, Gatsby Computational Neuroscience Unit
2013–current: Google Brain Team, Engineering Fellow
2016–current: Vector Institute, Chief Scientific Advisor

■Biographical Sketch

Dr. Geoffrey Hinton has been researching artificial neural networks for 47 years. Working with David Rumelhart and Ronald Williams, he showed that deep neural networks can learn useful distributed representations of complex data. Working with Terry Sejnowski, he invented Boltzmann machines.
His other contributions include mixtures of experts, variational inference and learning for neural networks, Deep Belief Nets and Capsule Nets. His research group at the University of Toronto used deep learning to revolutionize speech recognition in 2009 and object classification in 2012.

■Honors and Awards

1998: Fellow of the Royal Society
2001: David E. Rumelhart Prize in Cognitive Science
2005: IJCAI Award for Research Excellence in Artificial Intelligence
2011: Herzberg Canada Gold Medal for Science and Engineering
2016: Foreign Member of American National Academy of Engineering
2016: IEEE/RSE James Clerk Maxwell Gold Medal
2016: BBVA Foundation of Knowledge Award, Information and Communication Technologies
2016: NEC C&C Award
2018: ACM Turing Award (with Yann LeCun and Yoshua Bengio)

■Major Publications

A Learning Algorithm for Boltzmann Machines: (with Ackley, D. H. & Sejnowski, T. J.), Cognitive Science, Elsevier, 9 (1): 147–169, 1985

Learning Representations by Back-Propagating Errors: (with Rumelhart, D. E. & Williams, R.J.) Nature 323 (6088): 533–536, 9 October 1986

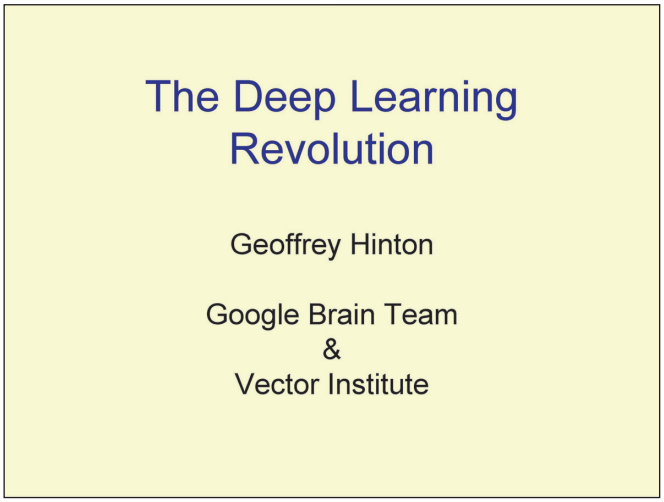
A Fast Learning Algorithm for Deep Belief Nets: (with Osindero, S. & Teh, Y.) Neural Computation, 18 (7): 1527-1554, July 2006

Reducing the Dimensionality of Data with Neural Networks: (with Salakhutdinov, R.R.) Science, 313 (5786): 504–507, 28 July 2006

ImageNet Classification with Deep Convolutional Neural Networks: (with Krizhevsky, A. & Sutskever, I.), NIPS 2012. Curran Associates Inc.: 1097–1105, 3 December 2012

This report is the gist of the commemorative lecture at the 40th Honda Prize Award Ceremony at the Imperial Hotel, Tokyo on 18th November 2019.

I would like to start by thanking the Honda Foundation and the selection committee for this great honor. I would also like to thank the many distinguished guests who have come to this ceremony.



I'm going to talk about the history of deep learning. I'm not going to talk about the most recent achievements in this area, and because I'm really a professor, I'm going to focus on actually explaining how it works.

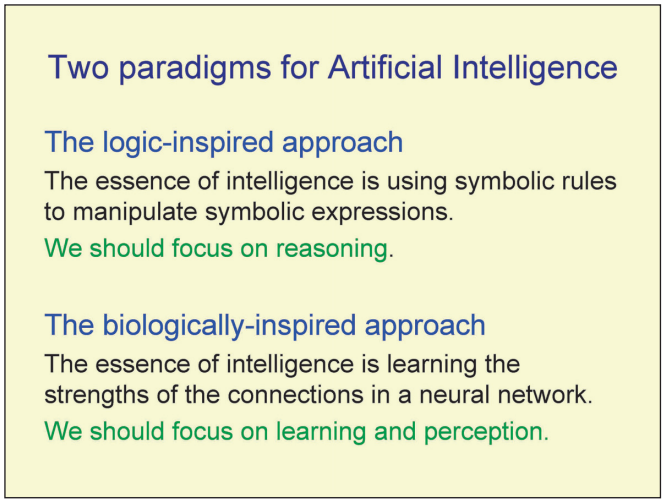


Fig. 1

〈Fig. 1〉 There are two paradigms for artificial intelligence. There is the logic-inspired approach where you think of intelligence as basically reasoning, and the essence of intelligence is manipulating symbolic expressions using rules of inference. And then there is the biologically-inspired approach, which is very different. In the biologically-inspired approach you think of the essence of intelligence as learning, learning the strength of connections in a neural network, and the main focus is not on reasoning, but on learning and perception.

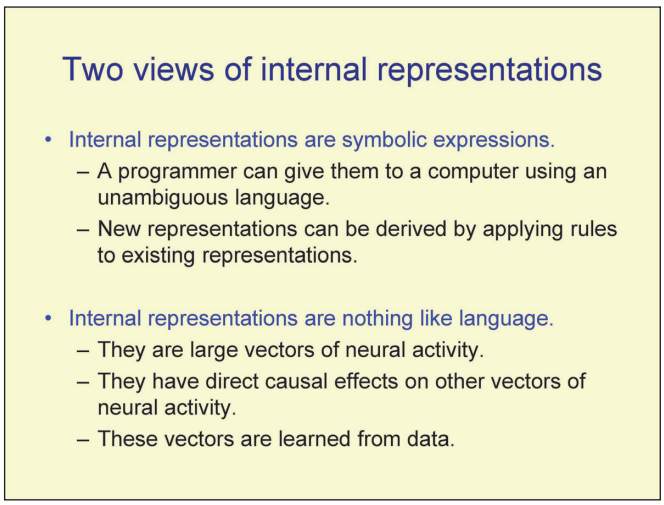


Fig. 2

〈Fig. 2〉 These two views of artificial intelligence go together with very different views of what internal representations are like. So in the symbolic approach, people think of internal representations as like strings of symbols. They are the kinds of things that a programmer can put into computer and they are the kind of representations that can be operated on by rules to derive new representations. In the biological approach, the internal representations are nothing like language. The idea is that inside your brain there are big patterns of neural activity. Language is just the way of getting stuff into the brain and the way of getting stuff out of the brain, but inside the brain it is not language at all, it is these large vectors of neural activity and these vectors have direct causal effects on other vectors, and the vectors are learned from data rather than being put in by a programmer.

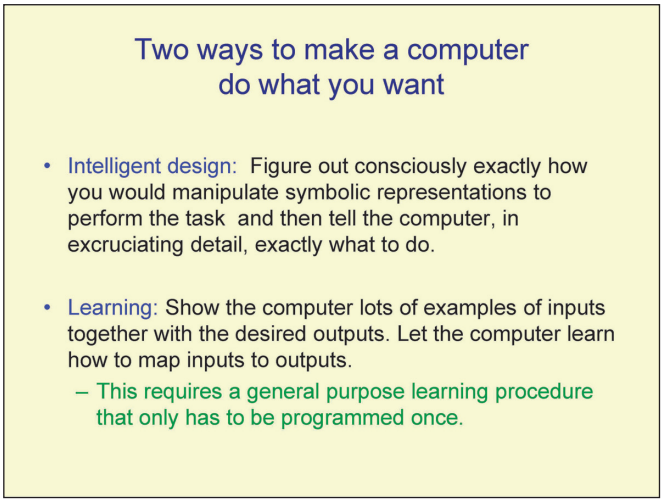


Fig. 3

〈Fig. 3〉 These two approaches lead to two quite different ways of making a computer do what you want. There is the approach I call intelligent design where an AI professor decides what representations the computer should have and puts them in by hand and figures out exactly how you would manipulate these expressions in order to get some task done and you

tell the computer in great detail how to do that. That is called programming.

There is a completely different approach where instead of programming the computer, you program it once to tell it how to learn and then for each particular task you just show it examples and it uses its learning algorithm to figure out how to derive the output from the input. So you've programmed it with this general-purpose learning procedure, which is back-propagation in the things I'm going to talk about, and for any particular task like speech recognition or object recognition it uses this general algorithm, but when you apply it to different data it learns to solve these different tasks.

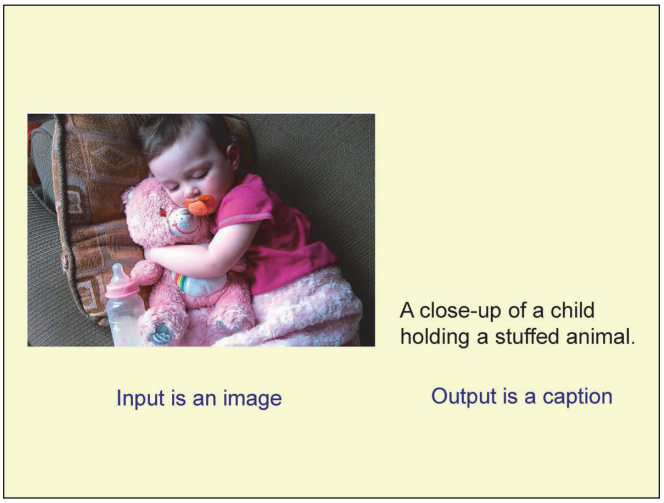


Fig. 4

〈Fig. 4〉 A good example of a task that was very hard to solve with symbolic AI was you give the computer an input, which is an image, so all the computer is really seeing is the red, green, and blue values of maybe a million pixels and so those numbers are coming into the computer, and what you want to come out of the computer given all these pixel intensities is a description of what is in the image, a string of words that says what is in the image. So for this image, a good string of words would be "a close-up of a child holding a stuffed animal." And that string of words was actually produced by a computer.

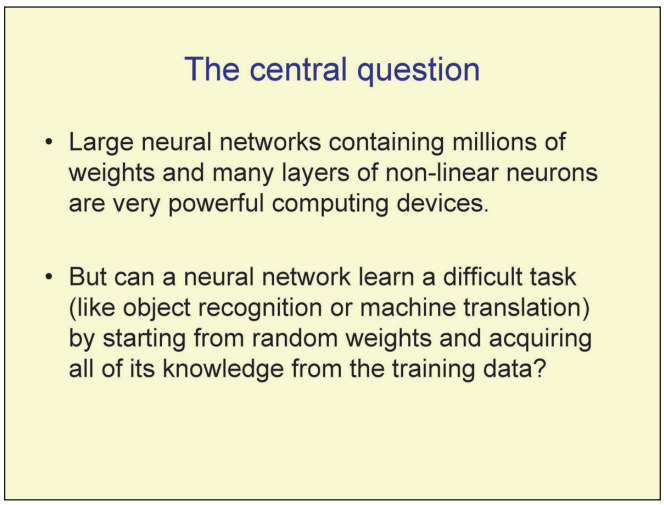


Fig. 5

〈Fig. 5〉 The central question for neural networks is can a large neural network that contains millions of connections, each of which has a weight that can be adapted, and many layers of neurons between the input and the output, it's a very powerful learning device but can it learn to solve difficult problems like converting an image into a string of words that describes the image just by starting with random weights and using a simple learning algorithm? And for many people that seemed completely ridiculous. They thought there is no way you're going to get a big neural network to solve a difficult problem like that if you don't tell it a lot about how to solve the problem. And we do have to tell it some things about how to solve the problem but basically these networks learn for themselves how to solve these problems.

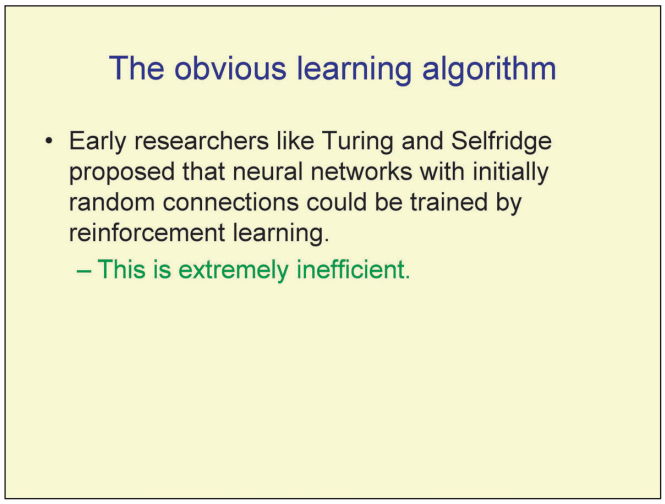


Fig. 6

〈Fig. 6〉 The obvious learning algorithm for a big neural network, which is obvious if you know about evolution, and it was obvious to researchers like Turing and Oliver Selfridge, was to start with a network with random weights and then to change one of the weights and see if it makes the network behave better. So you give the network some examples of inputs and

you look to see what outputs it produces, and then you change one weight and you see if it works a bit better.

Now, that's a very inefficient algorithm because you have to feed an example through the network, which involves using all the connections, and actually you have to feed maybe hundreds of examples through the network, see how well it does, then change one weight and now if that improves things you've learned how to make one small change to one weight. So this algorithm will work, in the end it will produce a network that works well, but it is extremely inefficient.

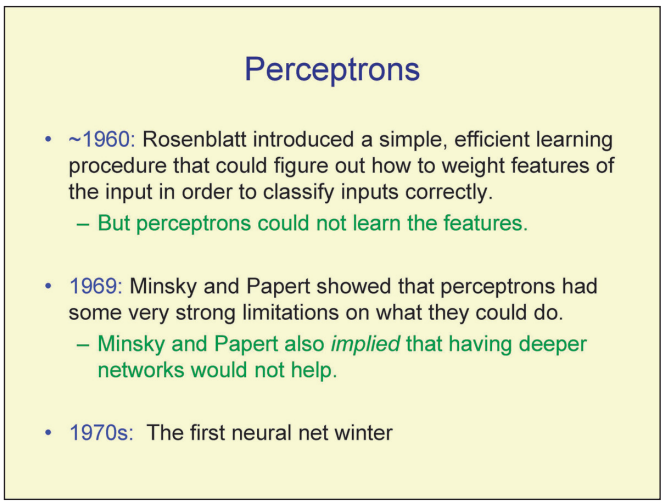


Fig. 7

〈Fig. 7〉 In about 1960, Rosenblatt introduced a much more efficient learning algorithm that works for simple neural nets. So what the learning algorithm learns to do is to decide how much weight to put on various features. You have some features that have to be designed by a programmer, but now the neural network decides how much weight to put on each feature in order to make a decision.

In 1969, Minsky and Papert showed that this kind of neural net is very limited in what it can do. It was a relatively efficient learning algorithm, but there were all sorts of things you can't do if you just have one layer of hand-designed features. You need many layers of features and Minsky and Papert didn't prove you couldn't do things with many layers of features, but since nobody knew how to train them it looked like neural networks were dead and we got the first neural network winter.

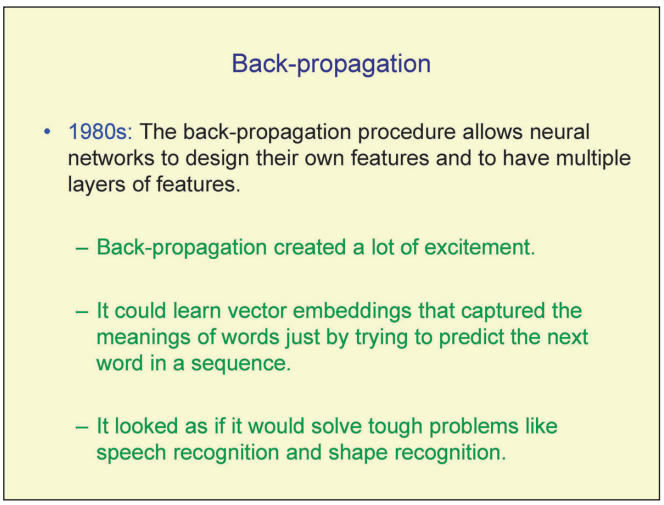


Fig. 8

〈Fig. 8〉 Then in the 1980s, many different groups developed the back-propagation algorithm. I was fortunate to work with David Rumelhart who was one of the people who thought of the back-propagation algorithm. And it allowed neural networks to learn layers of features, so it was a way of adjusting the weights in the neural network so it would learn many layers of features, and this created a lot of excitement as Amari-san (Dr. Shun-ichi Amari, Honorary Science Advisor of RIKEN Center for Brain Science) pointed out in his previous speech.

We showed early on that it could take strings of symbols and learn to convert each symbol into a vector so that from the vectors representing the first two symbols in a string you could predict the vector representing the third symbol in the string. So that was an example of a neural network learning to convert symbolic input into internal vectors. We thought at that point we would be able to solve many problems this way, and currently we can solve problems this way, but back then it didn't work very well and we didn't really know why.

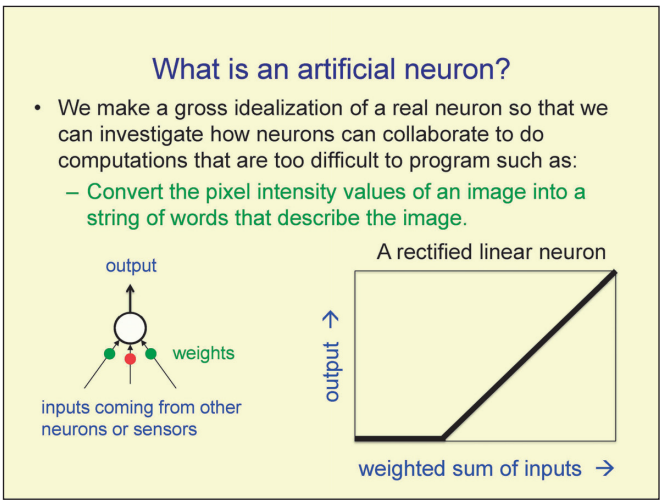


Fig. 9

〈Fig. 9〉 I am going to explain in a bit more detail what the back-propagation algorithm is

and how it works. And first I have to explain what an artificial neuron is. So the neurons in the brain are very complicated devices. What we wanted to do was simplify them and focus on the problem of how they interact to solve difficult problems. And, of course, when you do that it is dangerous, when you simplify something you lose some of the details, but the crucial thing is to simplify it in a way that allows you to study it without losing too many details. And so that was the hope of artificial neural networks.

Now, to begin with what we did was we used a particular kind of neuron called a sigmoid neuron and what it does is it takes inputs coming from other neurons or from senses and on each input it has a weight, so the weights are indicated by these dots here, and they adapt to make the neuron behave differently and the neuron gives an output.
And after about 20 years of trying one kind of neuron, we tried a different kind of neuron, this didn't happen until about 2010, and this different kind of neuron worked considerably better. And that kind of neuron has, if it gets input that is below its threshold, it gives an output of zero, but as soon as it gets above its threshold it gives an output that increases linearly as it gets more input.

So what you do with a neuron like this is you take the activity on an input line, you multiply it by the weight on the connection, you add all that up, then you put it through this function. So if you get a big sum, you'll get a big output. If you get a small sum, you will probably get an output of zero. That's the basic computing element.

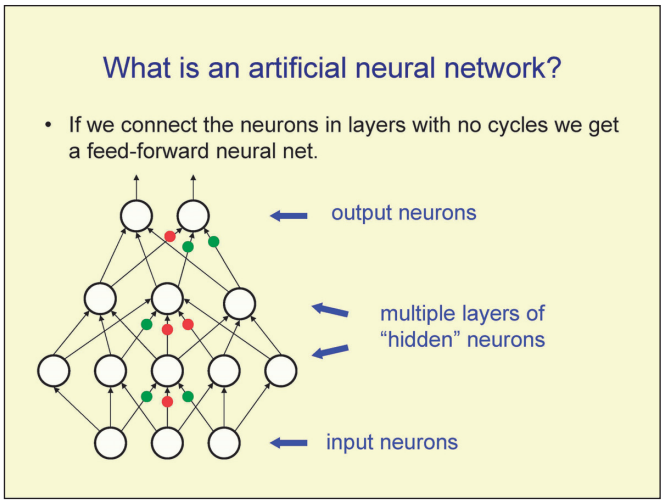


Fig. 10

〈Fig. 10〉 We then take those elements and we arrange them in a network. The simplest kind of network is a feed-forward network. So here we have multiple layers of neurons. These are the inputs, so they might be pixel intensities. These are the outputs, so they might be the probability with which you think this image belongs to a particular class. You might, for

example, be trying to decide whether something is a cat or a dog and this might be cat and this might be dog. There are weights on all of the connections and so what happens when you put in an image is you get certain activity levels in these neurons that depend on the weights, and as you change these weights what you are doing is changing the circumstances in which this neuron will be active and that means you are changing what kind of feature detector it is.

If you wanted to recognize a bird, for example, you might want neurons in the first layer that recognize straight lines, and neurons in the second layer that recognize straight lines that join at a fine angle, and if you see two neurons like that together, that are two straight lines joining at a fine angle, that is some evidence for the beak of a bird. If you then see that as well as that there are some neurons that represent edges that form a closed circle like an eye, and then the right spatial relationship, then that is a good cue for recognizing a bird.

So the idea of these nets is you will learn these connections that will turn these neurons in the intermediate layers into feature detectors, the feature detectors will detect more and more complicated features as you go through the network, so this might be edges and this might be something like a beak, and then when you get enough evidence of these more complex features they will activate whole classes. Now, of course, the neural networks we use are much deeper and more complicated than that, but that is the basic principle.

And the question is, how do you learn all of these connection strengths? So I told you the very simple algorithm. The very simple algorithm is you start off with random weights here, you try it on a bunch of examples, you see how well it predicts the right answer, and then you change one of the weights.

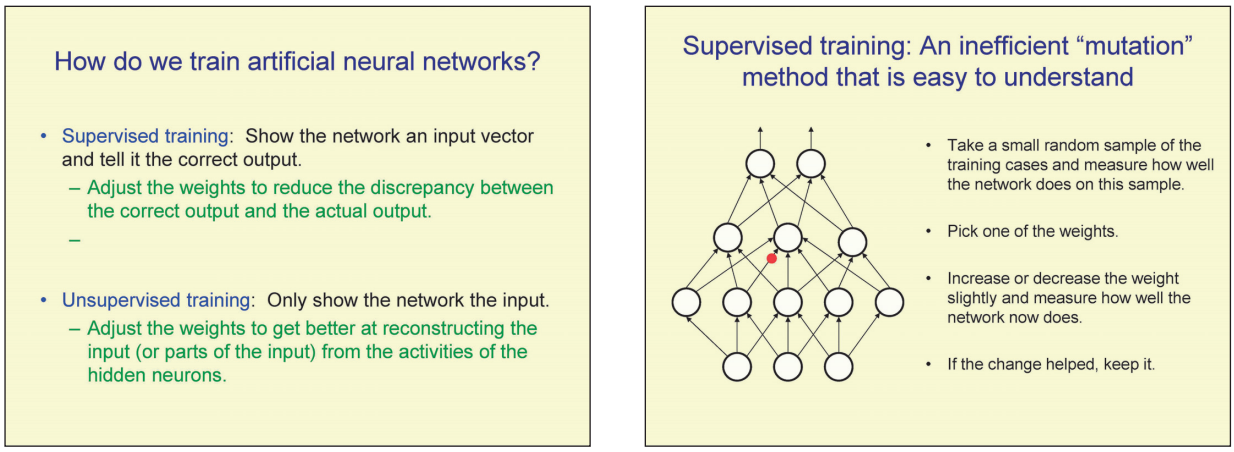


Fig. 11

〈Fig. 11〉 So, for example, you might change this weight here and then see if the network behaves better or worse. If it behaves better, you keep that change and then you try a whole

bunch more examples and you change one more weight. And you have to change each of the weights, and you have to change each of the weights many times, so you end up having to put billions of examples through the network, and since you only live for about a billion seconds that can't be how people work, there isn't time.

So what the back-propagation algorithm is, it is a way of figuring out how changing each of these connection strengths will improve your answer, but figuring out for all of the connection strengths in the network at the same time.

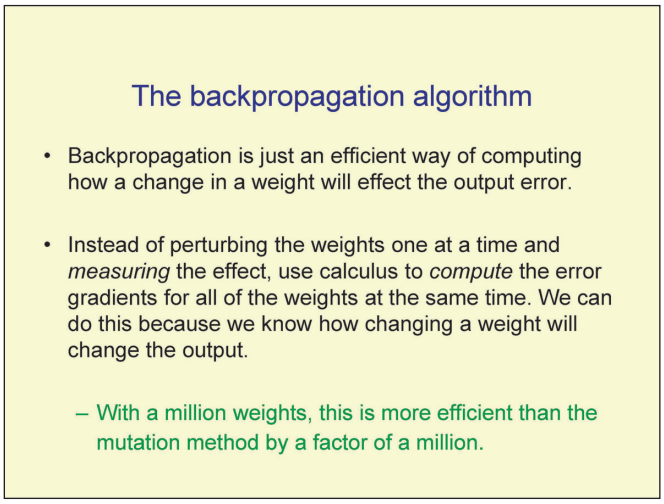


Fig. 12

〈Fig. 12〉 Instead of changing a connection strength and measuring the effect, which is like an evolutionary approach, what you do is you do a forward pass through the network and then having done the forward pass you do a backward pass through the network and that sends information backwards telling every weight in the network how the change in that weight would change the error. So you're trying to compute a derivative, which is how fast the error changes as you change that weight. And the point of back-propagation is you can compute all those derivatives at the same time for every weight in the network. So now if you have got a network with a million weights, you've got an algorithm that is a million times more efficient than the evolutionary algorithm, than the simple mutation algorithm, and that makes a huge difference.

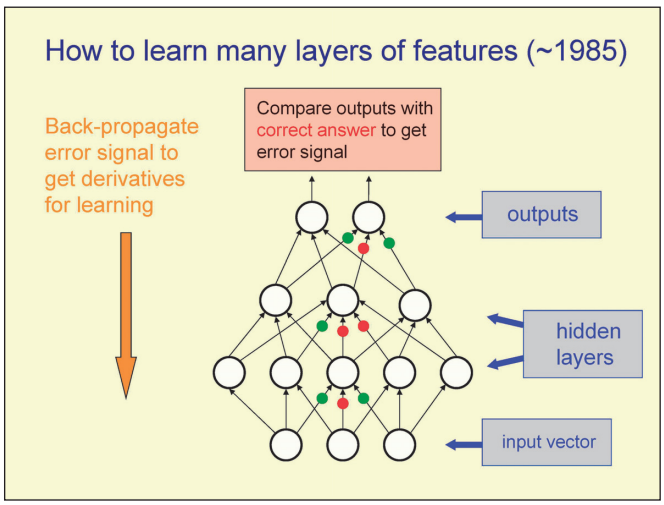


Fig. 13

〈Fig. 13〉 So in back-propagation, you would take some input, you would put it through these hidden layers with the current values of the weights, that would give you some probabilities of the different answers, and you would compare those probabilities with the correct answer. For this input it might be cats so this is the correct answer, for some other input it might be dogs so that would be the correct answer. Then you take the discrepancy between the answer it gives and the answer you would like it to give, and then you send signals backwards through the same network using the same weights, and by doing that you can compute for every weight at the same time how changing that weight would improve things, and then you change every weight a little bit in the direction that would improve the performance of the network. And now if you repeatedly do that many times, the network will become very good at doing whatever it is you want it to do, if you make the network big enough.

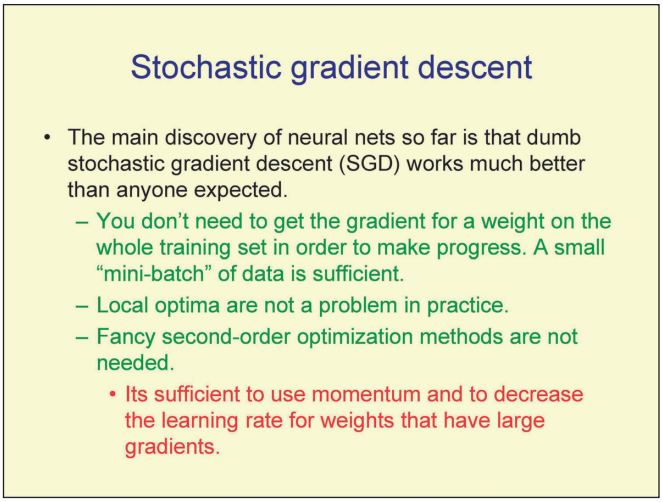


Fig. 14

〈Fig. 14〉 So once you've got the gradient that is computed by back-propagation, that is for every weight in the network you have computed how changing that connection strength

would improve the performance, you now update the weight in the direction that will improve performance and you do that for all of the weights in parallel, but you do it in order to increase the performance of the network on some small batch of training samples.

So you don't have time to put all of the possible training samples through the network. You just pick a small random batch of examples, you train the network to be better on that batch, which might make it worse on other examples, but now you take another random batch, and a very surprising result is that if you keep doing that the network will become very good at all of the examples. You might have thought that you'd have to compute the gradient on all the examples, that would be the standard optimization method.
But what people doing neural networks discovered was is that it is much more efficient if you have a big training set. Just take a few examples, use those examples to decide how to change the weights, then take a few more examples and change the weights again. And if you change the weights in a bad direction because you had untypical examples, that will make things temporally worse on other examples, but when you get to another batch you will cancel that out and change them in the right direction again.

So stochastic gradient descent actually works extremely well and so at that point we thought we had a very powerful learning algorithm, and in fact we did, and so we tried it on all sorts of things and it worked pretty well but it didn't work a whole lot better than other learning algorithms. There were other learning algorithms that worked just as well, and so that was a huge disappointment.

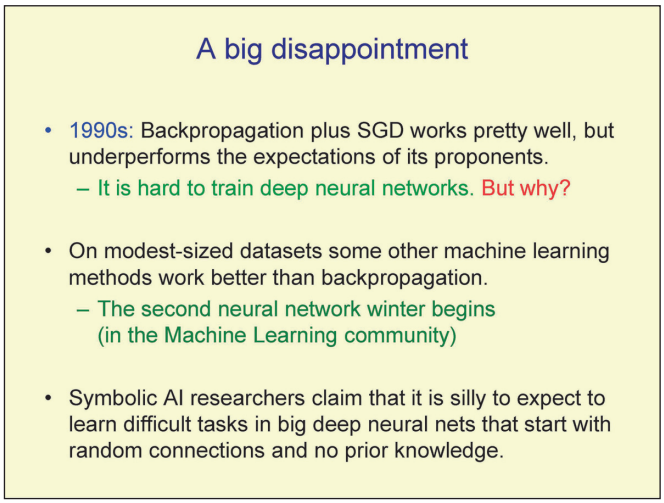


Fig. 15

〈Fig. 15〉 And at the time we didn't really understand that the problem was that this technique only really works well when you have big data sets and big networks. Using small data sets and small networks, it doesn't work better than other learning techniques, so it's

something that really comes into its own at scale. And back in the 80s we couldn't achieve that scale. We didn't have big enough sets of data to train it on, and we didn't have powerful enough computers to train it.

What happened was the people doing neural network research made all these claims that we are going to be able to solve really tough problems now, but then we couldn't solve these really tough problems. We could sort of solve them but not really well. The people doing symbolic AI, of course said well that's because you have got this crazy belief that you are going to learn everything from scratch. You are going to take a neural network with random weights and learn everything and really what you need to do is program knowledge into it. The only way you are going to be able to solve difficult problems is by programming in knowledge. Neural network researchers like me didn't believe that but we couldn't prove it was wrong because we couldn't make it work on really difficult problems.

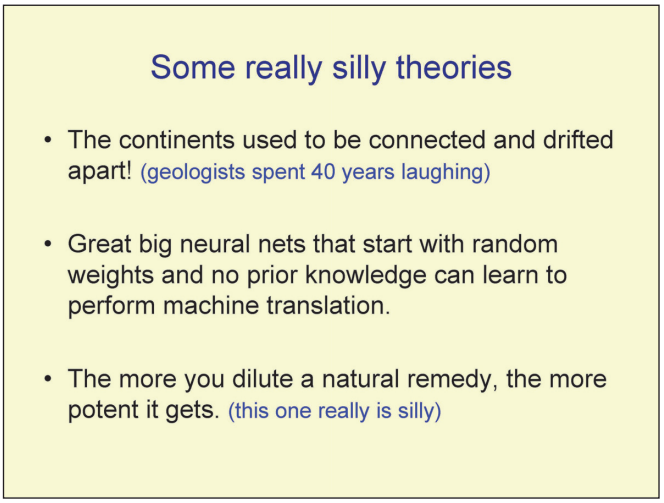


Fig. 16

〈Fig. 16〉 At this point I want to just describe some other really silly theories because you can get comfort when people say your theory is silly from looking at other silly theories. So one of the silliest theories that was proposed by Wegener in the 1920s was that the continents all used to be connected together and then they drifted apart. Now, he had some very good evidence for that.

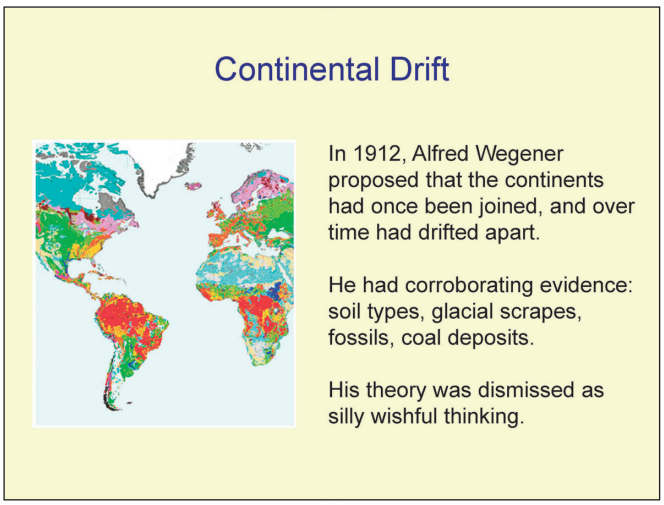


Fig. 17

〈Fig. 17〉 If you look at the coast of South America here, and you look at the coast of Africa, and you look at the soil and the rocks, they fit together quite well. Similarly, up here you find similar rocks here to here in Norway and Canada. You also find other things. You find that the fossils, very old fossils, that you find here are very like the fossils you find there, and the fossils you find here are very like the fossils you find there. And there's no real explanation for why you should get one kind of thing here and here and another kind of thing here and here, except that these used to be connected and when those animals lived they were living in the same place and then they drifted apart.

Wegener had lots of other evidence. He had evidence that in the tropics you have rock formations that have big scrapes in them. They are the kind of scrapes that can only be caused by boulders being pulled along by a glacier. So he had incontrovertible evidence there were glaciers in the tropics. He also had evidence that there were coal deposits in the Arctic. This doesn't make any sense if the earth is stationary.

The only way to explain this data is that the continents moved apart, but geologists had this theory that the earth was rigid and so they completely dismissed his view of continental drift and for 40 years geologists treated this as complete nonsense.

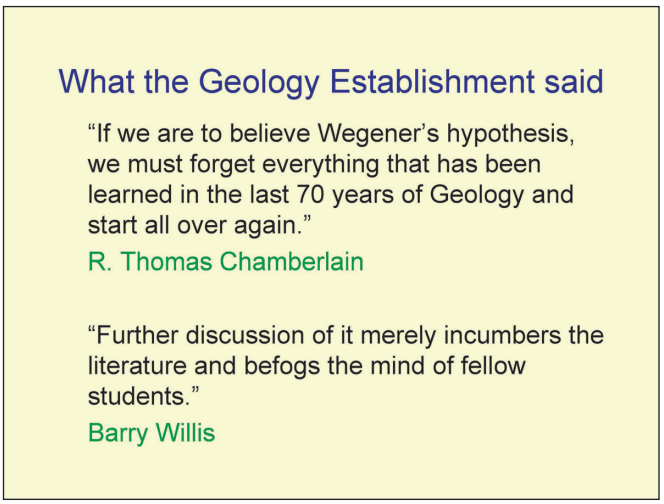


Fig. 18

〈Fig. 18〉 Here's a couple of things the geology establishment said. I took comfort from this because when I was a child my father was involved in the debate over continental drift, and based on biology he was on the side of continental drift because you couldn't explain the fossils otherwise. And so I saw when I was very young a theory that was dismissed as very foolish actually later being proved to be true and I think that probably had an influence on me.

So people said things like if you believe Wegener we have to throw away everything we know about geology, which turned out to be true. And they said things like we should not allow this stuff into the textbooks because it will confuse students.

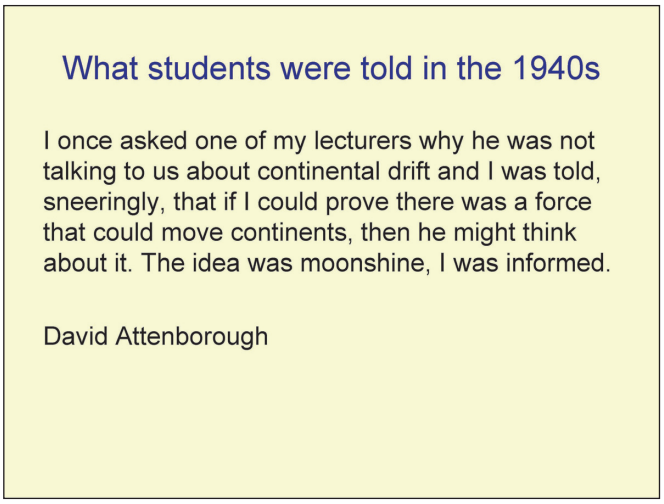


Fig. 19

〈Fig. 19〉 There is someone you probably know called David Attenborough who has made all these wonderful series about animals on the planet. When he was a student he was interested in continental drift and he was told it was complete nonsense because there was nothing that could make continents move. Now, actually people had already proposed the mechanism for

moving continents. The mechanism was that there was hot magma and the continents floated on the hot magma, and that had already been proposed in the 1930s, but geologists were so convinced the earth was rigid, that was the overwhelming view, that they wouldn't take this theory seriously.

Similarly, within AI, people were so convinced that you had to program knowledge in that they wouldn't take seriously the idea that you might actually learn it all. And they had some reason for disbelieving that because we hadn't shown that it worked yet, so once we began to show that it worked there was still a lot of resistance.

At the beginning of this century there were many examples of people producing impressive scientific results that were just dismissed by the scientific community as nonsense, and I'm going to just name three of them. I'll only do this for one slide. It's a self-indulgence.

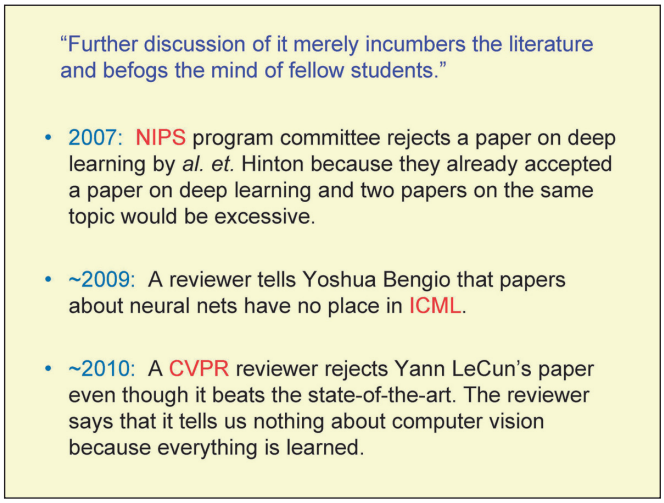


Fig. 20

〈Fig. 20〉 This is taken from the continental drift literature. That is what someone said in the continental drift literature.

In 2007, with a student I submitted a paper to NIPS on deep learning, one of the first two papers submitted to NIPS on deep learning. And it was rejected by the conference and I asked the conference why they rejected it and they said another paper on deep learning was submitted by Yoshua Bengio and that they couldn't accept two papers on deep learning in the same conference. That would be too many. Now, if you go to that same conference, most of the papers are on deep learning.

In 2009, Yoshua Bengio submitted a neural net paper to a machine learning conference, one of the main machine learning conferences, and the paper was rejected and a reviewer said it

shouldn't even be refereed because neural networks had no place in a machine learning conference. The community was so confident this stuff was nonsense that they didn't even want to review it for their conference.

Even worse, in 2010, Yann LeCun submitted a paper using deep convolutional neural networks for doing image segmentation and it beat the state-of-the-art, so it worked better than anything that the computer vision researchers had produced, and the paper was rejected. And the reason the paper was rejected was because everything was learned and so the computer vision community said that if everything is learned, it is not telling us anything about computer vision. They had already decided that the solution to computer vision was to program in lots of knowledge, and so the only question for computer vision is what knowledge you program in. And this work by Yann LeCun, even through it worked better, didn't tell you what knowledge to program in so it shouldn't be accepted by the conference, even though it worked better. They didn't understand that the whole point of the paper was it learned everything, you didn't need to program knowledge in.

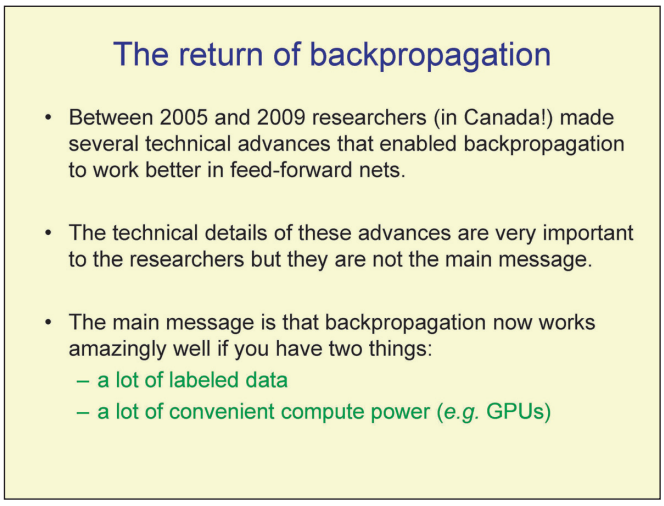


Fig. 21

〈Fig. 21〉 So between about 2005 and 2009, researchers, particularly researchers in Canada like myself and Yoshua Bengio, and Yann LeCun who we count as an honorary Canadian because he's French, made several technical advances that enabled back-propagation to work better. In particular, we found better ways of initializing the weights. So they start off random and then you do some unsupervised learning to initialize them before you do back-propagation and that makes the whole system learn much more easily. And the main result of that work is that back-propagation now works very well and it works very well if you have a lot of labeled data and you have a very fast computer.

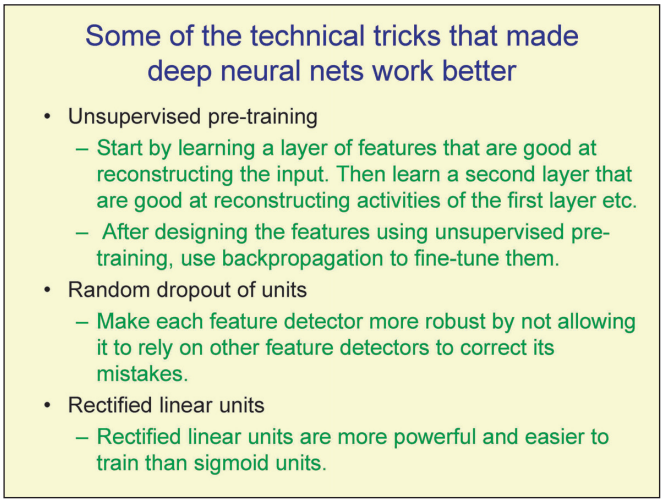


Fig. 22

〈Fig. 22〉 Some of the technical tricks that we introduced were to do unsupervised pre-training. If you are learning layers of features, you can learn them by having a deep network and back-propagating error from the output, but when the weights are initially random that's not the best way to learn. A more effective way to learn initially is to say I'll learn one layer of features at a time and I will try and learn features that allow me to reconstruct the input from those features. So, your first layer of features are learned so that from the feature activities you can reconstruct the pixel intensities. And then your next layer of features are learned so that from the feature activities in the second layer you can reconstruct the feature activities in the first layer and so on. And you can learn many layers of features that way, and when you've initialized the features like that it is then much easier to train the network with back-propagation.

We also introduced a technique called dropout where each time you use a neural network you randomly remove a random subset of the neurons. It turns out this makes it slower to learn but it makes it generalize much better. Basically, it stops the neurons relying on the other neurons. The neurons have to become a bit more independent from one another, they have to individually find useful features instead of relying on other neurons to correct the errors.

And the last thing I've already mentioned, we switched from using sigmoid units, which had been used for years, to using rectified linear units, which are those ones that if the input is small you get zero output, and then as soon as the input is above the threshold the output is proportional to the input. That's a sufficient nonlinearity.

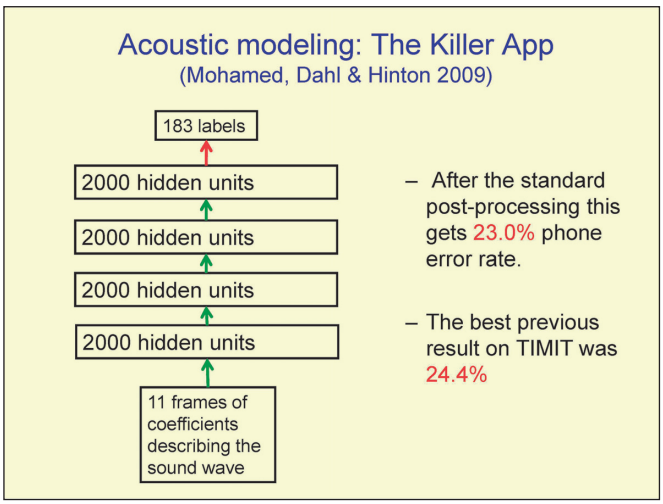


Fig. 23

〈Fig. 23〉 Once we had those things in place, two students in my lab applied these kinds of techniques to acoustic modeling. In an old-fashioned speech recognizer the way it would work is you would take the speech wave, you would use Fourier analysis or something like Fourier analysis, to figure out how much energy there is at each frequency and you would get something like a spectrogram. You would then look at the middle of the spectrogram and ask the question which part of which phoneme is the speaker trying to express by this pattern in the middle part of the spectrogram? So you have the context of the rest of the spectrogram and you are just trying to decide which piece of which phoneme the person is saying. And it turned out that we can make neural nets do that a little bit better than the previous technology to begin with and then later on much better.

And so to begin with, these students developed a neural network that would get frames of coefficients here describing the sound wave, they would then have multiple layers of these feature detectors that started of with random weights and turned into interesting feature detectors, and then they would have 183 labels for 183 pieces of phonemes that you might be expressing. That is called an acoustic model because it goes from the sound wave to fragments of phonemes. It is then up to some other model to decide which fragments to believe. You try and string them together into something that is plausible English.

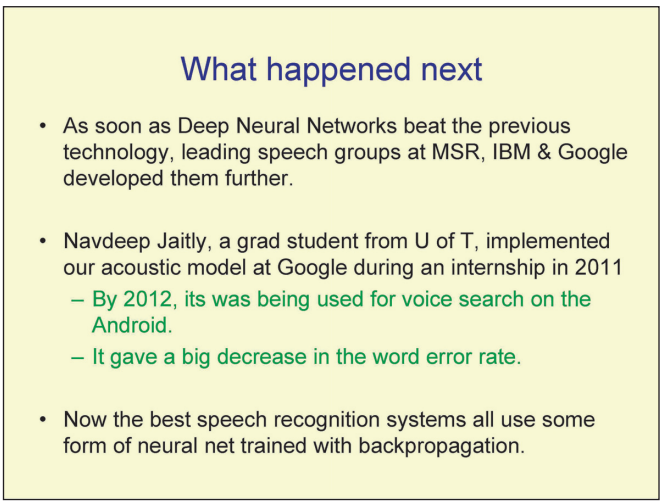


Fig. 24

〈Fig. 24〉 And so to begin with, neural nets like this worked better than the alternatives and my students took them to many different labs and showed different labs how to use them. And they went to IBM, they went to Microsoft, one student also went to Google, and the student who went to Google managed to take the acoustic model developed in Toronto and incorporate it in Google's speech recognizer and get a significant improvement. So in 2012, the research done at Toronto with just some engineering changes to make it more efficient, came out in Google voice recognition, Google voice search, on Android, and it worked noticeably better than Siri, and so that convinced all of the big speech teams that they should be using neural networks.

And now all of the best speech recognizers use neural networks but they don't just use them for the front-end that converts sound waves into bets about what piece of a phoneme it is, the whole system is now just a big neural network. You put sound waves in one end, and out the other end you get the transcription, you get the characters of the transcription, and there is nothing but neural networks and it is all trained end-to-end.

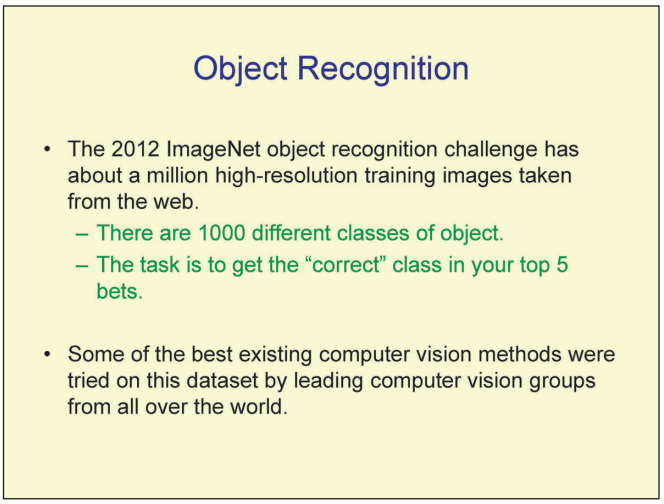


Fig. 25

〈Fig. 25〉 Then in 2012, at about the same time as the speech recognition came out on

Android and convinced big companies that this stuff really works in products, two more of my students Alex Krizhevsky and Ilya Sutskever entered the ImageNet object-recognition competition and this was finally a competition that had a big enough training set so neural networks could really work. Up until that point most object-recognition competitions had had a small training set and that meant that these learning techniques couldn't compete with hand-programmed techniques. There just wasn't enough data. But once we got enough data, they could do very impressive things.

And in this competition there was a test set that wasn't public. The people who ran this competition had this private test set so you couldn't cheat by making an algorithm work for the test data. And they would run your algorithm on the test data and see how well it worked. You would give them your network or your algorithm and they would run it on the test data to see how well it worked.

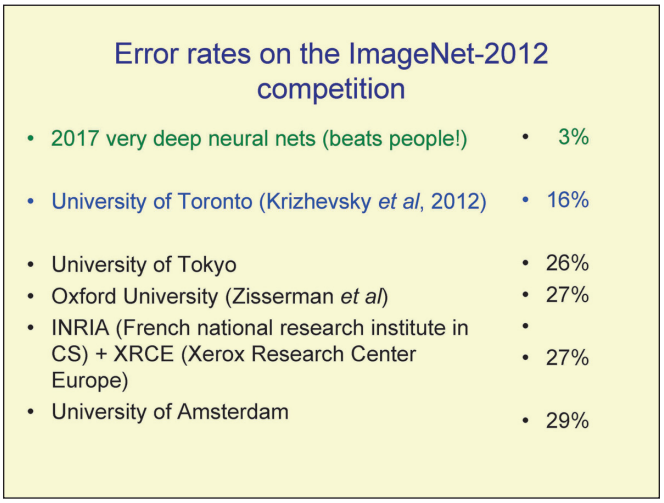


Fig. 26

〈Fig. 26〉 And the results of that competition were that actually the best conventional object recognizer was done at the University of Tokyo. It was slightly better than one done at Oxford and again slightly better than one done by a collaboration of the French National Research Institute and the Xerox Research Centre Europe, and what you notice about the conventional techniques is that they are asymptoting at about 25% errors. What Alex and Ilya, with some help from me, did was get something that only had 16% errors and that was a dramatic improvement. And that really convinced the vision community that these techniques actually really did work. And then the vision community, which a couple years earlier had rejected Yann's paper because it was using learning, they behaved like a scientific community ought to behave, they said, "Hey, this stuff works better than what we were doing. We are all going to change to using this stuff." Not all of them, and it took a couple years, but in only a couple of years the community changed completely and now anybody doing object recognition would

always choose to use neural networks, particularly convolutional neural networks.

This was the breakthrough that we got in 2012 getting from 26% to 16%, that was a big reduction. Since then, people have developed neural networks much further and there has been a big community effort developing them and now they are down to 3%. And humans on this dataset get about 5% errors, so on this dataset they are better than humans now. In general, they are not better than humans. On specific data sets they are though. So for particular problems like identifying skin cancer, they are now as good as the best humans and they will soon be better. For interpreting CAT scans, they will soon be better than the best humans. Anywhere where there is very specialized data and you have got a lot of it, then these convolutional neural networks are going to make huge improvements.

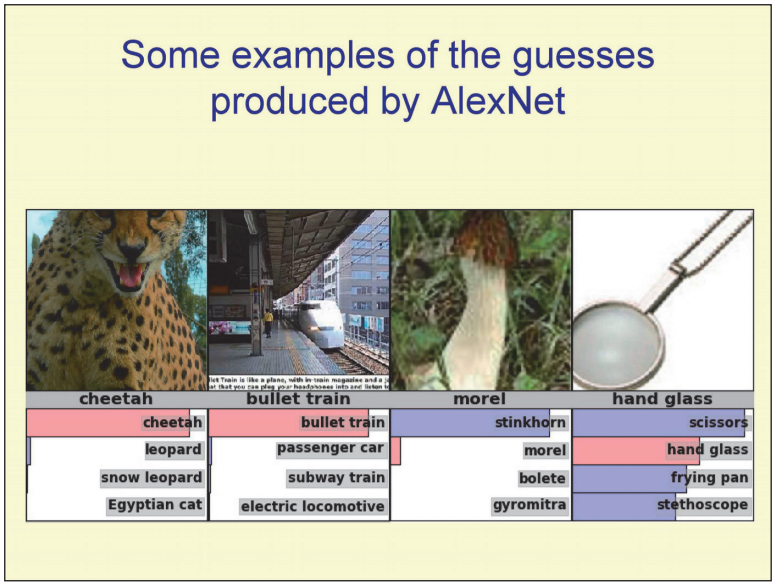


Fig. 27

〈Fig. 27〉 Here are some of the examples from the ImageNet competition. So what is interesting about the images is that they were taken from the web and they are not the sort of typical views of things. This is a picture of a cheetah from rather close-up, perhaps too close-up, and the pink is for the answer given by the neural network. In the ImageNet competition you are allowed to give five answers and you are counted as correct if one of your five answers was the answer that a person gave. What you will notice is its first answer is the correct answer, but its other answers are all pretty plausible answers.

Again, here it says it is a bullet train but notice the bullet train is only a small fraction of the image. There is a building that is much bigger than the bullet train and there is a person here. It has sort of had to learn things like on the whole the thing you should label is the thing in the middle of the image and that bullet trains are interesting. Its other alternatives are fairly reasonable things like electric locomotive or even passenger car.

Here it is very good at discriminating different kinds of mushroom, much better than I am, and it gets this one right. Sorry, it gets this one second. It is a morel and it thinks it is a stinkhorn.

This is an example of an image from a catalog, and you can see that its first answer is scissors, which is not the right answer. You can see why it might think it is scissors because this looks a bit like handles of scissors and this looks a bit like the blades of scissors. You could also see why it might think it is a frying pan. What is happening is its vision is not that good but the errors it makes shows you that it is really sort of understanding visual concepts. So it makes very plausible errors.

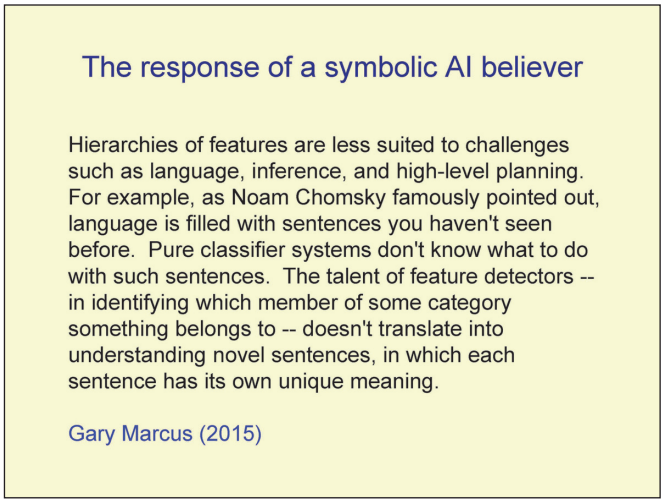


Fig. 28

〈Fig. 28〉 So once we had shown that neural networks trained with back-propagation, big neural networks trained on big data sets, could actually learn to do speech recognition very well and could learn to do object recognition, the symbolic AI people responded. Some of them changed their minds but a true believer in symbolic AI is someone called Gary Marcus, who publishes a lot, and in one of his chapters that he published in 2015 he claimed that this kind of learning algorithm might be fine for developing features for doing things like recognizing classes, but this kind of algorithm is going to be no good for dealing with language because it is not going to be able to deal with novel sentences.

There's an old argument going back to Chomsky who really didn't believe in learning, that because language is full of novel sentences you can't use statistical techniques to deal with language. This argument is completely wrong but it was very prevalent in the symbolic AI community and Gary Marcus still appears to believe it. So the idea was that because you are going to have to deal with novel sentences, these feature detectors are not going to be any good.

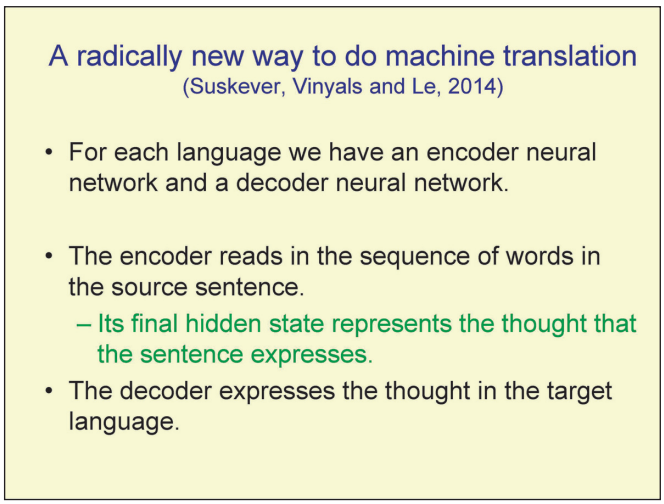


Fig. 29

〈Fig. 29〉 Well, a year before this was published the people at Google and in Yoshua Bengio's lab in Montréal had actually shown that there is a radically new way to do machine translation that can deal with novel sentences, and in about 2015 it was about comparable with the other methods of doing machine translation. Now the neural net way of doing it is much better than the other methods and is what Google Translate uses and that is why Google Translate now works a lot better than it used to five years ago.

And the idea is that for each language, the idea originally was, it has got a lot more complicated than this now, for each language we will have a recurrent neural network that reads words one at a time and converts these words into an internal vector. And then once we have read a sentence, we have converted the string of words into a big vector inside the network, and that vector is what I call a "thought vector." It is a vector of neural activities that represents the thought expressed by the sentence.

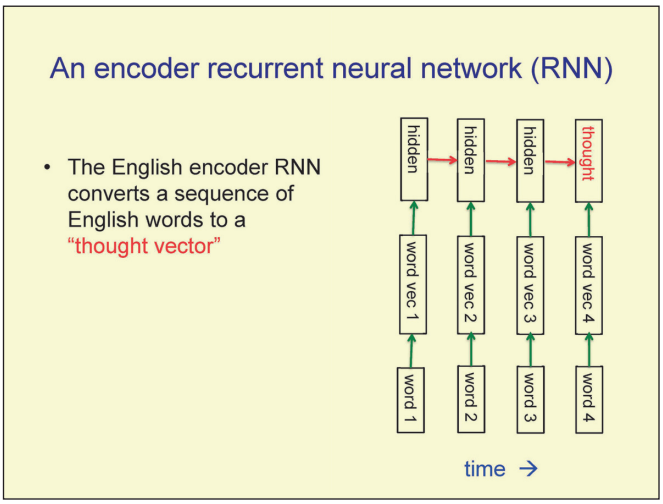


Fig. 30

〈Fig. 30〉 So the encoder would work like this. You put in a word, this will be a symbol. It

will convert that symbol into a vector, and that is a big bunch of neural activities. That vector would then provide input to other neurons here, and you get a vector of neural activities here that represents that the first word of the sentence was this. You then take the second word, you convert that into a vector, and now the input here is the information coming from the second word plus the information that you've already got in the sentence, and so now you will have a vector that represents a sentence where you know the first two words. And you keep doing this until you get to the end of the sentence and by the time you get to the end of the sentence you have got a vector here that represents, I should say only if all the connection strengths are right, but once you've learned all these connection strengths, and I will explain how you do that in a minute, this vector will represent the thought behind the sentence and it will represent it for one particular language. If you want to deal with a different language, you train a different network to deal with that language. So the English network will know a lot about the order of words in English and how adjectives come before nouns. The French network might learn something different. So this is the way you convert a sentence in one language into a thought.

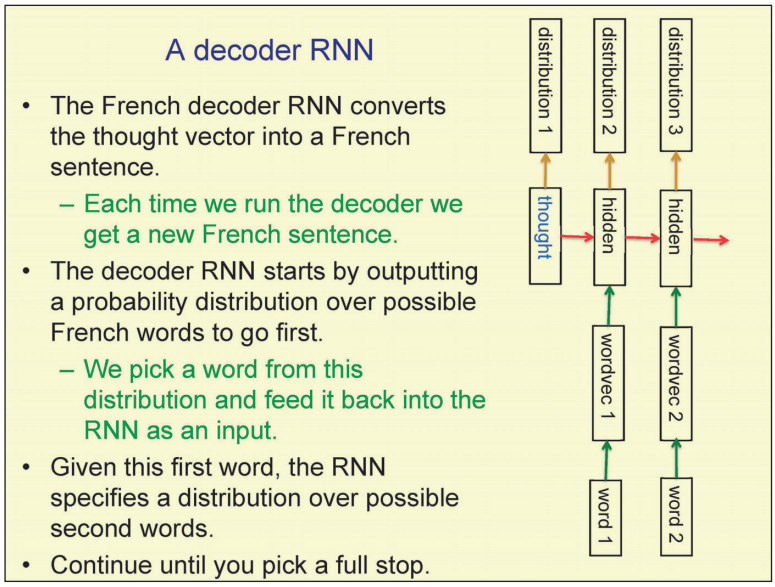


Fig. 31

〈Fig. 31〉 What you then do is you take the thought and you convert it into a sentence in the output language. So you might have a French decoder and what the French decoder would do is it would start with a thought and it would say given this thought what do I think the first word of the sentence might be? And it will have probabilities for the first word. It might think with a probability of 40% the first word is "le" and with a probability of 30% the first word is "la" and with a probability of 10% the first word is "chat" and so on. And so this probabilistic network says about what the probabilities of the words are.

And if you want to produce a translation what you do is you pick one of those words according to those probabilities. So if it thinks “le” is very likely you probably are going to pick “le.” And so you pick “le” and then you put “le” back in as the input here, it turns “le” into a vector, it then uses that vector combined with this thought to say, “Okay, if I’ve got this thought and I’ve just said the word ‘le,’ now what else do I need to express?” And that is going to be represented here, and then it chooses the rest of the sentence, what it needs to express, and now decide what word to produce next. And it decides probabilities for these various words, it picks a word according to those probabilities, feeds it back in and at this point it knows the first two words of the sentence. So maybe it is “le chat,” and given that it produces a representation here that knows that you have already said “le chat” and this is the thought you were trying to produce so what should you say next and so on.

And the way you train these two networks is you stick them together and to begin with they are random weights so it produces nonsense in French. It produces strings of words that are not translations of the English. But for the English sentence you give it, you know what the correct French translation is and so you say, let’s suppose that at this point it said there is a very low probability of producing the word “chat,” but actually in the translation the correct word to produce was “chat.” So you say, “Okay, I’m going to change all these connections and all these connections and all these connections so as to increase the probability that at this point it will produce the word “chat,” and that is just back-propagation through this network to figure out how to change the weights. So you increase the probability of “chat,” but you also figure out how you would change the thought vector here to increase the probability of “chat.”

And once you know how you want to change the thought vector, what you can do is you can go back to the encoder network and you can say, “Okay, I know how I want this thought to change so I can figure out how to change all these weights and all these weights and all these weights so as to change the thought so it will produce the right translation in French. And you just keep doing that for millions of pairs of English sentences and French sentences and if you start off with pure random weights in the network, after you have done this for long enough it can now translate.

Now, that system by itself produces not very good translations but not too bad either. It will translate. Since then there has been a lot of research on how to make it work better. I’m not going to go into that research.

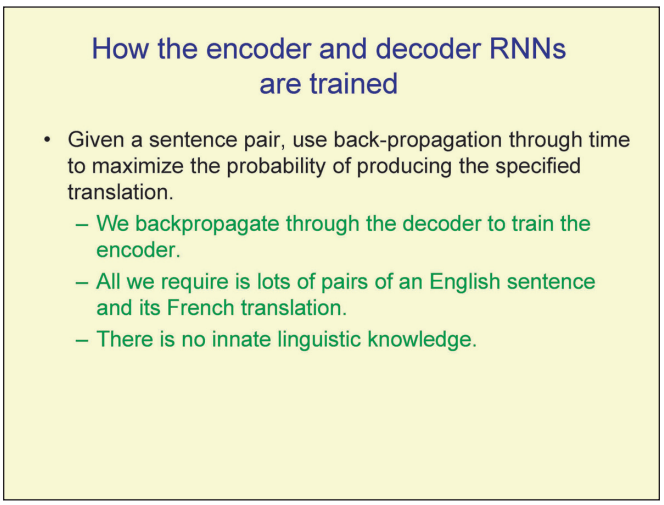


Fig. 32

〈Fig. 32〉 The main point of this demonstration was this is a relatively simple way of training a network with back-propagation to translate from one language to another, and the network does not need any linguistic knowledge. You just feed words in in one language, you tell it to produce words in the other language, you start with random weights, and the amount of linguistic knowledge you need to program into the network is zero.

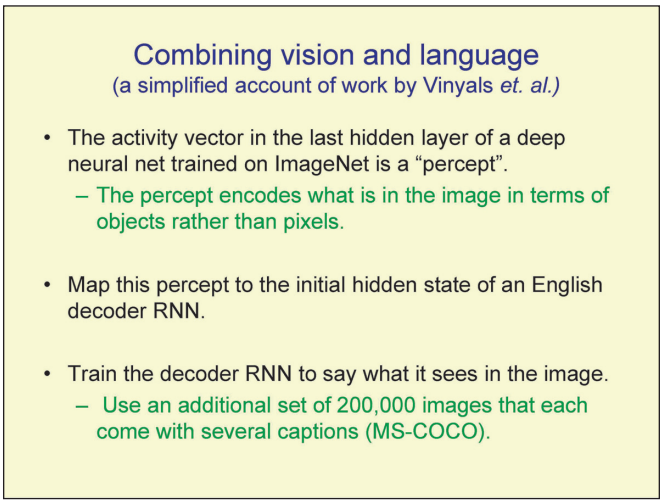


Fig. 33

〈Fig. 33〉 Now I want to go back to the example I showed you at the beginning, of an image and a caption for the image, and show you how we can now solve that problem. So we solved it by putting together the machine translation system and the ImageNet recognition system.

First you take the image and you put it into the ImageNet recognition system and at the end of the system you have bets about the different object categories, but just before the end, the last layer before the end, you have a big vector of neural activity that has taken the pixels of the image and turned them into information about the kinds of objects that might be in the image and you treat that big vector as a percept. And you take that percept and now you take

your language decoder and instead of putting in the thought you got from something that encoded English into a thought, you put in the percept and you say take this percept and say what this percept is in language. To do that you need a training set and Microsoft provided a very nice training set where you have images and you have captions. So you have already got a system that can recognize the images, you take the last layer of neurons in that system, you treat them as input to the language decoder, and you just tell it to say that and it learns the weights so it can say what is in the image. And it works.

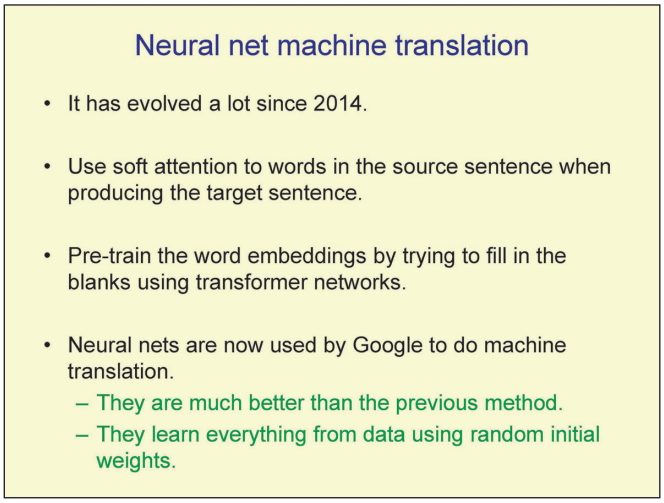


Fig. 34

〈Fig. 34〉 Neural net machine translation has actually evolved a lot on since 2014 when we first got it working with this simple idea of turn a string of words in one language into a thought and then express the thought in the other language, and one big factor has been attention. As any translator knows, you don't listen to the whole of the sentence in the first language and then say the sentence in the second language without thinking back to the first language. So now what happens is as you are producing the translation, you are looking back at the words in the first language, or rather their hidden representations, to influence what you say and you are deciding which bits of the input sentence you should attend to as you are producing the output sentence. That makes things work much better.

And one other thing that makes things work much better is actually doing pre-training. So instead of training everything in order to translate, you actually take strings of words in one language and what you do is you leave out some of the words and get the network to reconstruct the complete sentence, even though you are giving it a sentence with words left out. It actually works even better if you leave out little sets of contiguous words, so you are leaving a little gap in the sentence. And what the network has to learn to do is fill in the gap but without translating it, just in English fill in the gap, and once you have trained it to do that then you train it to do machine translation, and by learning to fill in these gaps, it learns very good vectors for representing the words.

For example, if you give it a sentence that has the word “may” in it in English. When you turn “may” into a vector, you don’t know whether it’s the month “May” or it’s the modal “may” as in “he may do that.” So what happened originally was the vector that you used for representing “may” was a compromise between a vector that represents the month and a vector that represents the modal. Fortunately, big high-dimensional vectors can actually be quite close to the month and quite close to the modal without being too close to other things so it worked. But now what happens is you feed in the words, you get a vector for each word, and then these vectors look at the vectors for neighboring words. And maybe in the same sentence a nearby word is “June.” So if you’ve got “May” and a nearby word is “June,” that helps you disambiguate “May” and think that it is the month. But if a nearby word is “might” or “could,” it will disambiguate it and you will think “may” is the modal. And that way of disambiguating words has made these language models work much better now.

Now they work really well and in fact they work so well that neural nets can now be trained on a big set of language not to do machine translation but just to predict the next word. And after you have trained them, you can give them one sentence to start off with and then you can get them to predict the next word, and whatever they predict you tell them they were correct and you feed that back in and then they predict the next word and you tell them they were correct, and these big networks now produce really impressive stories, they produce stories with long-term coherence.
You can use exactly the same technique for producing music and now neural networks can produce music that actually sounds like real music. It’s quite difficult to know whether it was composed by a neural network or a person.

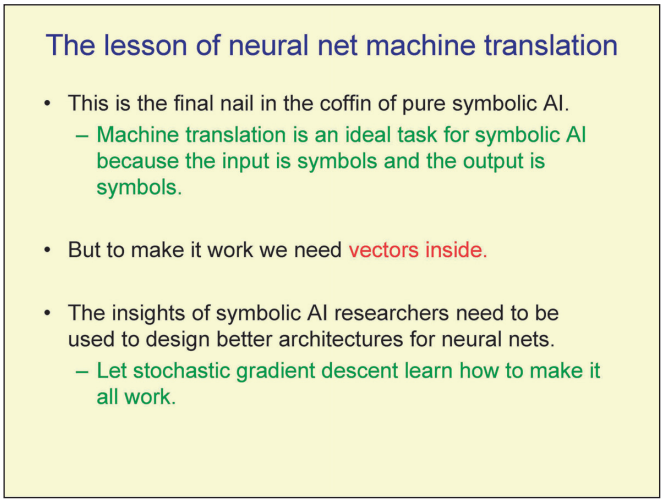


Fig. 35

〈Fig. 35〉 So I think the lesson of neural net machine translation was that for machine translation it’s the problem that ought to be best suited to symbolic AI because the input is a

string of symbols, words in one language, and the output is a stream of symbols, words in the other language, and the obvious way to try and do machine translation is to manipulate the input symbols to produce the output symbols. So you might have thought what you wanted was rules for manipulating symbol strings symbol strings and that is how translation used to work, but actually that's not the technique that works best. You want to take the input symbols and you want to convert them into great big vectors of neural activity and then use interactions among those vectors in order to produce the output symbols.
So at the input it is symbols, at the output it is symbols, but inside it is all vectors and it is vectors having causal effects on other vectors.

Now, I don't think symbolic AI was a waste of time. People doing symbolic AI had a lot of insights into how people do reasoning but they tried to implement those insights on conventional computers, which is not the right way to implement them. What we need to do now is take the insights of people who did symbolic AI and use those insights to help us design better neural networks. But we need the learning algorithms of neural networks to do the heavy lifting to actually decide what all the weights should be rather than trying to program everything by hand.

So I think it's fairly conclusive now that if you want to build an intelligent system for recognizing speech or recognizing objects or for translation, you ought to do it with great big neural networks that start with random weights. They may be structured in various ways that depend on insights into how you need to solve the problem, but basically they learn everything from the data.

So this is the end of my talk.

HONDA FOUNDATION

Editor in chief　Akihiro Kameoka

Tel. 03-3274-5125　　Fax. 03-3274-5103

6-20, Yaesu 2-chome, Chuo-ku, Tokyo 104-0028 Japan

Tel. +81 3 3274-5125　　Fax. +81 3 3274-5103

https://www.hondafoundation.jp