Week 4: Key Applications — Computer Vision and Natural Language Processing

Video Transcripts

Video 1: What Does Computer Vision Study?

PIETER ABBEEL: In this session, we'll study computer vision, which strives to give computers the ability to see.

A bit more precisely, here are the goals for this session. First, build an understanding of the types of problems computer vision studies, and I say studies because, indeed, computer vision is still much a research field. But a research field that has progressed very fast over the past few years.

So, we'll build an appreciation for how fast this field has progressed over recent years and why this is likely to continue for years to come. We'll build an understanding of what underlies this recent fast progress and become familiar with key buzzwords. We'll make sure you know how to identify the current best-performing systems, their level of performance, and axes along which you want to measure performance.

So, what does computer vision study? First, classification. In a classification, the task is to recognize what's in an image. For example, for this image, we'd want the computer to conclude it's a cat.

Often, just deciding what's in an image is not enough. You also want to know where it is. Object Detection systems have put bounding boxes around classified objects, and instance segmentation does pixel-level segmentation of each object.

Images tend to be 2-D, but they are a reflection of the 3-D world all around us. 3-D reconstruction is concerned with reconstruction of the 3-D geometry of what's around us. Especially important for applications like self-driving cars, as pictured here.

While traditional computer vision has been largely about processing images given to the computer, lately, there has also been a lot of progress in automatically generating images that look like real images. For example, all images shown here, they might look like real images, but actually, they are not real images. Rather, they are all computer generated. And anytime we give the computer, really underneath a large neural network, a new random seed as input, it'll generate a new image as its output. Much like an artist can create new images from their imagination rather than needing to record directly from the real world. And, in fact, what you see here is a painting automatically generated by a deep neural network. And it was sold at Christie's for a lot of money.

Finally, as we progress in this lecture, we'll see a lot of advances in computer vision are achieved by annotating images with what's in each image. This is called providing supervision, and the learning happening is called "supervised" learning. However, annotating these images is very tedious. Semi-supervised learning tries to reduce the need for annotation by also learning from images that don't have any annotation.

The graph shown here is a very recent breakthrough, showing that, in blue, semi-supervised learning can now consistently outperform supervised learning, shown in red. So, indeed, we can now learn quite a bit faster when supplementing the annotated data with additional not annotated data. Now, before we dive into how computer vision might solve some of these problems, you might wonder, with so many research papers being published, all making progress on these topics, how can I go find the current best system?

Well, it turns out there is a webpage, paperswithcode.com, which keeps track of performance of latest systems on a very wide range of tasks, including in computer vision. The URL is included on the slide. So, let's take a look at what this website has to offer!

It benchmarks systems for semantic segmentation, for image classification, for object detection, for image generation, and many more. And for each of these tasks, it has evaluation performance across many instances of the task. So, if in your job, you need, let's say, an image classification system, this could be a natural starting point.

Video 2: Classification: Progress and Architecture

Alright. Now, let's dive into classification, how it's done. We'll look at recent progress, and we'll see that this progress is closely tied to advances in neural net architecture designs.

So, the big breakthrough that took computer vision from not working to becoming sufficiently capable to start building a wide range of applications was the ImageNet competition, organized by Fei-Fei Li.

The ImageNet competition has been held on a yearly basis since 2010. In this competition, you get access to 1.2 million images from a thousand object categories. And your task as a participant is to build a system that can recognize which object is in each image.

For example, it might be a mite, or a container ship, or a motor scooter, or a leopard, and so forth. The organizers of the competition have their own secret stash of 150,000 images. When you participate, you send your computer vision system to the organizers, and they then run it over their 150,000 secret images (called the "test set"), and they report out the number of errors your system made when trying to classify those secret images.

Here is a graph showing the results in this competition. On the horizontal axis, we see the years the competition was held. On the vertical axis, we have error rate, so the lower is better. We see the best entry in the competition in 2010 had about 30% error rate; it wasn't much better in 2011.

And progress of traditional computer vision approaches was flatlining. But then, in 2012, out of Geoff Hinton's lab at the University of Toronto, there was the first deep learning entry into the competition. And it cut the error rate in half! This was a huge surprise and a huge proof point for Deep Learning. Of course, people noticed, and they switched approaches. And we see here that there wasn't just a one-time leap forward. There was actually a massive acceleration in progress. And in fact, by now, this competition has been retired, as human-level error rates have been achieved on this benchmark.

Now, to get a better understanding of what powered all this progress, let's take a deeper, more detailed dive into what's behind this graph. As we already know, in 2010 and 2011, all entries were traditional computer vision approaches, which we can also think of as using a single-layer neural network, so very shallow neural nets.

Then, in 2012, Geoff Hinton, together with his students Alex Krizhevsky and Ilya Sutskever, won the competition with the first deep learning entry. Obviously, this was a major breakthrough in image classification. But it was also much more than that. It was arguably the biggest milestone in signaling the potential of deep learning to outperform classical methods. The neural network they submitted has become known as AlexNet, named after the first author of the paper, Alex Krizhevsky. So, what does AlexNet look like under the hood?

Here is the architecture drawing directly from the paper, and also a more schematic diagram on the right. AlexNet consists of five convolutional layers, where a local mask is passed over the image. Convolutional layers tend to do local processing, which makes sense, as understanding what's in an image has a lot to do with local pixel configurations.

But it's also good to keep in mind that after going through multiple convolutional layers, the information has started to diffuse out and isn't nearly as local anymore. And then, from there, AlexNet has three fully connected layers, which bring all of the information together to ultimately decide which of the 1,000 possible classifications to assign to the current image. And, actually, it doesn't output just one choice. It outputs a probability distribution over 1,000 possible classifications. We'll see this general structure come back quite a bit as we step through the progression on the ImageNet competition.

Now, the big AlexNet / ImageNet breakthrough happened in 2012. A very big deal in putting deep learning on the scene. But, actually, people had been working on deep learning for a long time. Admittedly, generally with less success. But there is a predecessor to AlexNet, called LeNet, named after Yann LeCun. Yann LeCun and collaborators established state-of-the-art digit recognition in 1989. The dataset is known as MNIST and often still used when wanting to run a very small, quick experiment on image recognition.

Here is the LeNet architecture. Just like AlexNet, LeNet already had multiple convolutional layers followed by fully connected layers. So, how come it took another 20 years before a similar architecture was successful on a more complex task like ImageNet recognition?

First, deep neural nets tend to excel in the regime where lots of data is available. ImageNet was the first such large-scale image recognition data set. Second, this took six days of training the neural network on two state-of-the-art GPUs at that time, so an experiment that would have been very time-consuming to run even just a few years earlier.

Alright, so we had our big Deep Learning breakthrough in 2012. What happened after? In 2013, Zeiler and Fergus won the competition with a neural network much like AlexNet, but they adjusted some of the hyperparameters. They made the convolutional filters smaller and also added more filters in each layer, which resulted in a 5% error reduction! In 2012 and 2013, the winning networks had eight layers. But the more layers, the more capacity the neural network has. So, in principle, the better it should be able to classify images.

In 2014, Simonyan and Zisserman achieved another significant reduction in error rate, now down to 7.3%, by introducing VGG Net. VGG Net was still a convolutional network, but now, 19 layers. One of the tricky things about VGG net was the number of parameters. 140 million parameters in the network had to be trained. One might wonder, is it possible to reduce the number of parameters while the network remains expressive enough?

Szegedy and collaborators from Google showed this is possible. In fact, they won the 2014 competition with a network with 22 layers but only 3 million parameters. How was this possible to have more layers, yet less parameters? They got rid of the fully connected layers, going convolutional all the way. And they also had some clever refactoring of the convolutional layers themselves.

Now, given that more layers seems to do better, what if we build even deeper networks? Turns out people at the time didn’t have much luck training deeper networks. The issue was that the optimization problem that underlies neural network training would become very poorly conditioned when having deeper networks, and deeper networks would actually do worse. This was until the next breakthrough happened.

ResNets! Kaiming He and collaborators at Microsoft brought the error rate down to 3.6% with a 152-layer network. Much much deeper than any previous network.

This 3.6%, by the way, is better than human-level performance on this benchmark! Humans achieve about 5% error rate. The key problem ResNets addressed was that, in principle, deeper models should be able to perform at least as well as shallower networks, but they didn’t due to vanishing gradient signals. The solution they proposed, add the option in the network to simply skip layers. This way, it’s very easy for the deeper network to propagate signals. Even more precisely, every residual block consists of two convolutional layers, which are additive to an identity layer. This way, an extra residual block in the network doesn’t have to learn the identity anymore but simply has to learn what’s needed in addition to what the previous layers already achieved.

From there, a few more architecture improvements were made, and error was driven down to 2.3% in 2017, at which point the official competition was retired, but the dataset itself remains an important, widely studied benchmark. Before moving on from this graph and the ImageNet competition, let’s zoom out for a moment. This graph shows some very important trends.

First, deep learning blew right past the performance of classical computer vision approaches. In fact, within a few years, achieved human-level error rates on this benchmark, something people absolutely didn’t expect would happen anytime soon, when asked back in 2010 or 2011. That’s when traditional computer vision methods were the mainstay and very limited progress was achieved.

Second, the deeper the network, the more expressive. Of course, the network must remain trainable. Key for this are the residual blocks, which consist of the identity plus a learned delta to the identity. These residual blocks remain the staple of most architectures today. And, related, deep remains key to learning powerful representations.

Third, quite a bit of human ingenuity went into designing these neural net architectures. Sometimes, they were fundamental changes, such as residual blocks. Sometimes, more hyperparameter tuning. Especially for the latter, one might wonder, couldn't we just write a program that searches over a range of architectures and see which one does best, then tries variations on that one, and repeat?

Such automation is actually what Google's AutoML tries to achieve. It's not fully there yet, and it's definitely fairly restricted as humans define the neural net architecture space for the AutoML to work with, but it's starting to become surprisingly competitive, as was showcased in a recent suite of ML competitions organized by Kaggle, where AutoML was able to do really well on a very large fraction of the competitions in that suite.

Video 3: Classification: Data Set Augmentation, Speed and Cost

So far, we've been talking about progress made by improving the neural net architectures. Another way to make progress, of course, is to get more labeled data. However, unfortunately, labeling data tends to be tedious, hence, costly. So, you might ask yourself the question. if we have a small pool of labeled data, can we use it to automatically generate a larger pool of labeled data? That's exactly what data set augmentation looks at.

For example, let's say you have this one labeled image that's a six. If we rotate the image a bit or, slant it, or make it narrower or wider, or make the pen stroke thicker or thinner, it'll still be a six!

Here is a table showing that very good automatic augmentation leads to outperforming state-of-the-art across many image recognition benchmarks. This particular AutoAugment algorithm was invented by Google and required an inordinate amount of compute. But since then, our research at Berkeley has made this a factor 1,000 less compute-intensive while matching the same performance.

Now that we understand that rapid progress that was made through deep neural net architectures and the importance of data augmentation, let's think about what other axes we might care about in image classification.

Imagine you or your team are tasked with setting up an image recognition system to solve a problem at your company. Certainly, you'll care a lot about accuracy, but likely, just as important will be the speed with which it operates and the cost associated with operating it. So, let's now take a look at those.

How to find how well various systems perform? There is this nice benchmark called DAWNBench, which considers all these axes other than accuracy. For example, you might be curious how long would it take to train a certain neural network and what kind of compute might I consider for training the network. This specific benchmark also considers ImageNet, and it shows that today, it's possible to train to 93% accuracy in less than three minutes. That's just phenomenal! Remember, AlexNet in 2012 took six days to train.

How about the training cost? It's now possible to train on ImageNet to 93% accuracy for just a little over $12 of cloud compute cost. How about inference latency? This refers to how long it takes from sending an image to the computer to getting back out the decision from the neural network as to what's in the image. We see the best systems can now do this in less than one millisecond. In this case, this is a 50-layer network, and everything gets computed through all 50 layers in less than one millisecond. How about inference cost? Well, rounded to the nearest cent, it's actually zero at this point!

Video 4: How to Run Image Recognition

Now that we've seen how well image recognition systems can work, you might wonder, where do people start, how do they run these? It turns out that, assuming your categories are fairly common use case. There are cloud services to which you can send an image, and you'll receive back an attempted classification of what's in the image.

However, it's good to keep in mind that none of these systems are anywhere near perfect. And, perhaps even more importantly, it's important to understand what these systems were trained on. Neural nets will do pretty well when the types of images they are trained on are the same as the types of images they get tested on. But if there is a mismatch between train and test, all bets are off!

What would be an example of a train-test mismatch? Let's say the system was trained to recognize cats versus dogs. But all images in the training data were taken indoors. Then, when presented with outdoor test images, it's quite likely the system won't perform all that well.

While image recognition of a service might be okay for some people's needs, it's probably much more common these days to still rely on training your own models on your own data to make sure you get the best possible performance for your problem.

Where to get started for that? Often, it's possible to just download open-source models from GitHub and then fine-tune them on your own data. You'll see that the frameworks used to train these models are mostly PyTorch (which is built by Facebook) and TensorFlow (which is built by Google).

Now, one question that will quickly come up when you have your own data: how to get it annotated? Since deep learning requires such a vast amount of annotated data to be successful, it's actually become a huge business in itself to provide data annotation services. Some examples of companies operating in this space are Scale.ai, who recently became the latest Silicon Valley unicorn; Figure Eight, which was acquired by Appen for use across many of their businesses; Mighty.ai, which was recently acquired by Uber so Uber could fully utilize their data annotation bandwidth; SuperAnnotate.ai, a more recent player, which I have personally been advising; Google Cloud AutoML, Amazon, and so forth. For all of these, essentially, you send off your images and a prescription on how they should be annotated, and they'll get them back to you annotated as you asked for.

Before wrapping up our discussion of image classification, I'd also like to call your attention to the medical domain, which has very different images, but the exact same ideas have been very powerful there, too.

For example, this Nature paper shows that a deep neural network can achieve dermatologist-level classification of skin cancer. And this paper shows that deep neural networks can achieve radiologist-level classification of whether or not a patient has pneumonia from the X-ray scan.

Video 5: Detection and Segmentation

Alright. We did our deep dive into classification. For the other four main thrusts in computer vision, we'll stay a little more at the surface, but under the hood, in fact, much the same is going on anyway. And with that, I mean iterating over fairly similar neural net architecture designs, being smart about data augmentation, and making decisions on how important accuracy is versus energy consumption, versus latency, and so forth.

Now, let's take a look at detection and segmentation. In classification, when given this image, a good answer could be sheep, or dirt road, or grass. In localization, it's not enough to just say what's in the image, but you also need to put a bounding box around where in the image.

In object detection, you can't just put a single bounding box around all the sheep. You need to put a separate bounding box around each individual sheep.

In semantic segmentation, bounding boxes aren't enough anymore. Now you are expected to annotate each pixel in the image with what's in it. So, in this case, road, sheep, and grass.

In Instance Segmentation, if there are multiple instances of an object, you need to individually annotate them.

Just like for classification, we can go to papers with code to see how well object detection systems work. Just like in classification, we see significant progress over the last few years, and we see that a 101-layer ResNext architecture comes in first. Similarly, for instance segmentation, there has been significant progress over the last few years, and we see that also here, a 101-layer ResNext architecture comes in first.

Neural net architectures for segmentation are a bit different than for classification. In classification, we are trying to bring all information from the image together, gradually extracting local information until finally, we know what's in the image as a whole and have just a classification output at the end.

But in segmentation, we need each pixel annotated with what's in that pixel. At the same time, we do want to use full image context similar to what's done in classification. So, here is an example of what this looks like the first half of the network looks a lot like a classification network, but then there is a whole second half, re-expanding to full image size. In this architecture here from 2015, the re-expansion is done simply by un-pooling operations.

These days, often people use what’s called a “U-net” architecture, where a skip connection is introduced from the mirrored layer on the other side. Intuitively, as the network processes the image starting on the top left here, working its way to the right and down, it’s building up more and more global understanding of what’s in the image, but at the expense of losing details. Then, when working back up to the top right, through the skip connections, it gets access again to the detailed information, which it can now fuse with the higher level understanding it’s built up coming from the bottom.

Video 6: 3-D Reconstruction: LiDAR

As humans, we don’t just semantically analyze what we see around us, we are also able to extract 3-D information, which allows us to much more fully understand the structure of the world around us, which, of course, is intrinsically 3D, even if images are 2-D. And this helps us to function in the world. So, how can we have a computer understand the 3-D structure of the world?

There are three main technology thrusts for 3-D reconstruction, often also referred to as depth perception: LiDAR, Stereo, and Monocular.

So how does LiDAR work? First, it sends a laser pulse, in this figure, down to earth. Then, it measures how long it takes for the laser pulse to come back after reflection of the surface. From this, we can compute how far away the surface is from the lidar unit. We simply need to calculate the product of the speed of light with how long it took, divided by two, because it had to go down and then also back up. This gives us one LiDAR measurement. Most LiDAR can send pulses in multiple directions and, in doing so, build up a depth map in all directions. And, of course, we can move the LiDAR unit around to map out other areas.

So, this all sounds great. Might it have any limitations? First, since measurements are done by multiplying, return time with speed of light, this requires a very precise clock to get precise readings, in practice, often can get around 2cm precision. Since LiDAR depends on sending light out and getting it back, typical units have only about 50m range (beyond that, would need to pump more power into them). Dark surfaces don’t reflect much light, hence, LiDAR struggles getting readings. Specular and semi-transparent surfaces might send the incoming laser beam elsewhere rather than back to the LiDAR unit.

Here is an example of a point cloud obtained from fly-over with a LiDAR. Note that we can see this point cloud from any viewpoint. It needn’t be the same viewpoint data was collected from. And here is another example, showing a 3-D point cloud collected from the car’s LiDAR but visualized from our view.

With self-driving cars deemed such a massive market, people talking about trillion-dollar market. And given that these cars have to see the world in 3-D, it’s not a surprise there is a lot of companies competing over building the LiDAR systems for those cars, a bunch of them listed here. Of course, their LiDARs can also be put to work for other use cases, for example, 3-D mapping, building or a city.

Video 7: 3-D Reconstruction: Stereo

So, we covered LiDAR. Now, let's take a look at stereo. How does it work? Triangulation! We have two cameras. We detect the same object in both cameras' images and can then triangulate where the object is in 3-D. In fact, humans are able to use their two eyes to extract depth information the same way.

What limitations might stereo have? Well, since it's impossible to perfectly pinpoint where an object is in the camera view. For one, we only have a finite number of pixels, and second, it might be hard to do the matching, we won't have those perfect green rays coming out that nicely intersect where the real object is. Instead, we'll have approximate rays, shown in blue here.

Now, if our cameras are very close together, which we call a narrow baseline, then these rays are running nearly in parallel, and a slight shift in localization of the point in the image will lead to a significant shift in where the rays intersect in 3-D. So, you might say, "Well, let's just use a wider baseline, so small errors in localizing x in the image won't have as much effect on where the rays intersect".

But new challenges arise, namely, the further apart the cameras are, the harder to match up points across images, as they start having very different views onto the scene. And, of course, there is also the physical aspect. It's easier to set up cameras closer together as one unit. It's also worth keeping in mind that stereo-matching, that is, finding which point in the left image here corresponds to which point in the right image, is very much 'not' a fully solved problem.

One way to simplify this problem, which is often done for indoor scenes, is to project patterns onto the scene with a projector. Those projected patterns are designed to make it easier to match points between the two images. Of course, this doesn't always work. For example, outdoors it's difficult because the projector has to compete with sunlight. Also, for dark, specular, or semi-transparent surfaces, the pattern might not be visible as hoped for.

Recently, there has been a lot of progress on matching points across two images with deep learning. The same kind of networks that are used to recognize what's in an image or to segment an image turns out they can be trained to detect which two points across the two images correspond to the same point in the 3-D world.

Video 8: 3-D Reconstruction: Monocular

So, we covered LiDAR and Stereo. Now, let's take a look at how we can extract depth from a single image. At first, this might seem a little too much to hope for. With a single image, it's not possible to triangulate across two images where a pixel came from in the 3-D world.

Indeed, depth from a single camera is geometrically under-constrained. In this example here, from a single image, it's not possible to geometrically determine what's exactly out there – would it be a smaller tree nearby or a bigger tree further away? But we know humans can perceive depth with one eye. In fact, you could cover one of your eyes right now, use the other eye to look around you, and you'll notice you still see the world as 3-D. How is this possible? The key insight here is that, while geometrically, the problem is under-constrained, we can still statistically make sense of it. For example, if we knew for a certain type of tree how tall it tends to be, then we would know how far away from us it's likely to be.

Here is another example. This here is a single image, snapshot on UC Berkeley's campus. Purely geometrically, we can't tell whether the tower at the top middle is behind the library, than what a tiny, tiny tower top that floated in mid-air right in front of the camera. But since we know the typical sizes, since we know that objects tend to be rooted on the ground, since we know how the sun will cast shadows, and so forth, we, as humans, can actually fairly reliable reconstruct the 3-D of this scene despite only having access to one image.

Computers have started to do this as well. Here is an example of this in action. This was one of the earlier results in this direction. These days, people would train large neural networks to predict the depth of each pixel. This is actually a very difficult task, but a lot of progress is being made.

Video 9: Image Generation Overview

Alright, so at this point, we covered classification, detection and segmentation, and 3-D reconstruction. What else can computer vision do for us? How about generating images?

Neural nets can take in very detailed descriptions of what you want and generate images that incarnate your text.

For example, let's ask for "A Victorian man struggles with his addiction to TikTok." Victorian men didn't have access to TikTok, of course, but let's see what we get. Here we go! I find this just amazing. The man is dressed like in the Victorian era, and he is clearly struggling. You can look at his face, and he is holding an alcohol flask. And even though we didn't ask for a phone, the AI just knows that TikTok is typically watched on your phone.

Let's do another one. "Darth Vader realizing he's forgotten to add an attachment to the email". We have all been there. Let's see what it got for us. Here we go. Amazing!

Let's do one more. "A photo of a Shiba Inu dog with a backpack riding a bike. It is wearing sunglasses and a beach hat." Here we go! This one is such a realistic image, except that dogs can't actually do this, of course.

How about image editing? Here we have a picture of "a vase of beautiful red roses". Let's now give it a correction, "the vase is blue". Let's see if it can make that update. Wow, I'd say beautifully done. So, we can now do image editing by simply talking to the AI.

You might wonder, how about video generation? Well, let's try that, "A sunset over a field of crops with the sun shining through the clouds and the sun shining through the leaves". Here it is. So, even video is starting to work with rather open-ended requests.

Let's try another one: "A dog crosses the street, runs into the park. Then the dog sees a ball and starts chasing it". Here it is. Okay. This isn't the greatest video you have ever watched, but it is starting to get the gist of this. Pretty amazing all the kinds of images and videos that AI can generate!

Video 10: Image Generation: PixelCNNs and GANs

You might wonder, how can a neural net generate such images and videos? How do we set it up for this, how do we train it?

Well, there are four main thrusts, and we'll cover all four because, as of today, it's not clear at all which thrust will long-term be more important. In fact, it is not unlikely that what will win out in the long term is a combination of them.

The first thrust centers around so-called PixelCNN models. The PixelCNN is a neural network that is trained to generate the next pixel when given all previous pixels in the image. For example, here, the PixelCNN neural net is supposed to predict the value of pixel x_i (in red) when given as input all the pixels preceding x_i in the image, shown in blue. So, the neural net is really set up to do something very similar as in classification, except that now, instead of outputting a label, it's outputting a pixel value.

Just like in classification, the neural net will not have a single deterministic prediction but rather a probability distribution over values the pixel could take on. Once done with training, we can generate images one pixel at a time, in each step sampling from this probability distribution over values for the next pixel.

A major downside of the PixelCNN is that by generating only one pixel at a time, the generation of a high-resolution image can take a long time.

PixelCNN is one line of work. A completely different line of work is Generative Adversarial Networks, which have the same goal, being able to generate new sample images.

The main idea in Generative Adversarial Networks (or GANs) is very simple. We want a neural network, called the generator network, to generate realistic images. For any random noise image we feed into it, it should output a different image that looks real. Well, if for every image it generates, we could give it a "realness" score, then that signal could be backpropagated to improve the generator until it generates maximally realistic images.

Now the question, of course, is, how to provide this realness score? Well, the brilliant idea here is that this realness score can be provided by another neural network called the discriminator, which is trained to distinguish real images from generated images. So, we are really training two neural networks at the same time here. A generator network trying to generate images that achieve high realness score, which in turn is evaluated by the discriminator network, which is trained on pairs of images, one real, one fake, and has to determine which one is real vs. fake.

It's worth noting that training data is used in a very different way here. The generator isn't directly trying to generate pixel-by-pixel what's in training images like for PixelCNN. Rather, the training images are used to train the discriminator network, which determines whether images are real or not. And it's this discriminator network that feeds back a "realness" score to the generator network, which trains against that to maximize realness.

Ultimately, what we have here is a two-player game. The generator tries to fool the discriminator into thinking the generated images are real. At the Nash equilibrium of this game, the generator generates realistic images, and the discriminator puts 50/50 probability on each generated ("fake") image for being real or not and does the same for real images.

How well might this work? Well, in 2014, when this line of work just started to emerge, here is what we got. Nothing super-realistic, but very promising. Then, in 2015, it had already gotten quite a bit better. By 2016, it'd gotten even more realistic. Even more so in 2017. And we got high-resolution, near indistinguishable from real faces since 2018. It doesn't have to be faces, of course. Here are some other examples generated by GANs.

The key drivers of this progress, more compute, allowing to train larger networks for longer, and architectural improvements.

Video 11: Image Generation: VAEs and Diffusion Models

The third main thrust in image generation is centered around Variational AutoEncoders or VAEs. The key idea here is that two networks are trained: An encoder network and a decoder network. The encoder network takes in an image, often referred to as x, and turns it into a code z. Then, the decoder is supposed to re-generate the original image x from the code z. To ensure this works, of course, the encoder has to generate a code z that contains enough info about the original image to allow the decoder to reconstruct it.

Now, there is an additional important thing that's done during training. Not only are encoder and decoder trained to reconstruct the original image, the training also ensures that when looking at the codes z corresponding to many images, that these codes z come from a distribution that's easy to sample from, for example, a Gaussian. By ensuring this, now, when asked to generate a new image, we can just sample a new z from the distribution p(z) and run the generator on that z to get the new image.

The so-called Vector Quantized VAE is capable of generating sharp, realistic-looking images, as shown here. Here are some more Vector Quantized VAE-generated images.

You might wonder, how does this compare to GANs? Right now, the quality of generation is very similar, with some people arguing the Vector Quantized VAE is better because, in addition to achieving the same image quality, the Vector Quantized VAE achieves better coverage, that is, it generates more variety in its images.

The fourth main thrust in image generation is centered around diffusion models. The idea behind diffusion models is actually pretty simple, but it turns out really, really powerful. During training, you take your images, we see one training image here on the right, and you gradually introduce noise into your image through a diffusion process. And then, you train your neural network to do denoising. In the end, the resulting model can take in pure noise and denoise it into a realistic image.

Many of our motivating examples were generated by diffusion models, which are currently probably the most popular. And actually, I am very proud to say that some of the research behind these diffusion models was done by me and my students here at Berkeley.

Video 12: Semi-Supervised Learning

The general idea behind semi-supervised learning starts from the observation that supervised learning requires a large amount of annotated data, which is often tedious and expensive to collect. Wouldn't it be nice if we didn't have to do as much data annotation, maybe by also being able to learn from data that's not annotated?

Well, that's exactly the question semi-supervised learning is concerned with! I'd also like to clarify here that semi-supervised learning is a very general discipline. It's not just done in computer vision. But here, we'll look at it in the context of computer vision.

To make things concrete, we will study a very specific instance of semi-supervised learning. We'll consider one large neural network, which still takes as input an image but that now has two output branches. The first output branch is the unsupervised output. It's an output that doesn't require any human annotation, and as a consequence, we can train the neural network along this branch on a very large number of images without requiring any annotation effort.

The second output branch is the supervised output. This output represents the task we ultimately want to solve. Training the neural net along this branch requires annotated data, but hopefully less so than if we didn't also have the other output.

Why might this work? Well, training mostly on the unsupervised output allows the neural net to learn an internal representation that reduces the need for labeled examples. Let's make this a bit more concrete.

Here is a network that takes in an image and has two outputs. One output is supposed to re-create the input image. The other output is the supervised one and is supposed to predict the correct label, in this case, the number "3".

Or another way to think of this; we have a Variational AutoEncoder along the top path. The Variational AutoEncoder learns a latent representation space z, where ideally, the data is represented in a semantically more meaningful way than raw pixel values, which then, in turn, should make it easier to train a classifier on the z space.

There are many ways people do semi-supervised learning. Here is another way. Here at the top branch in the neural network was trained to turn greyscale images into color images. It's very easy to get training data for this: just download color images, make them greyscale, and you can start training; no annotation effort required.

The thinking here is that a neural net that correctly colors images must have some semantic understanding of what's in the greyscale image. That semantic understanding is built up as the network processes the input image, working its way from left to right. At some point, we can branch off and feed into a second output branch network that is trained for classification.

Results obtained this way showed the resulting network requires less annotated data to learn to classify than a network that's trained only for classification.

It’s also a pretty cool result in itself. Here are some examples of the network turning greyscale images into color images. And here are some classical images that only exist in greyscale, and now the network has generated a color version for us.

Here is a third idea people have played with for semi-supervised learning. We still have the classification output, of course, but then, on the additional output, the network learns to predict whether the image has been rotated 90, 180, 270, or 0 degrees. Again, this is a prediction problem that requires no human annotation effort. You just download images, and then you can choose with what rotation to feed it into the network and train it to predict that rotation.

Turns out a network that can reliably predict such rotation is surprisingly ready to then learn to classify from just a small number of annotated images.

Finally, the most powerful extra training losses to date try to fill back in blanked-out regions. Masked AutoEncoders literally fill in the blank in the image, and there are also closely related approaches like Contrastive Predictive Coding that work in embedding space.

Let’s take a quantitative look at how much this can reduce the need for annotating images. On the horizontal axis, we have the number of annotated images used. On the vertical axis, we have accuracy. So higher is better. We see that as more annotated images are used, both approaches have improvements in accuracy, which is, of course, what we’d expect.

What’s so exciting here is the difference between the blue and the red curve. In red, we have one of the best supervised learning approaches. In blue, we have the semi-supervised learning approach using fill-the-blank. We see that, for example, when having access to only 12 labeled examples, semi-supervised achieves 65% accuracy, whereas supervised only achieves 43% accuracy. We also see that semi-supervised consistently outperforms supervised.

And naturally, in the limit of very large amounts of annotated data, in this case, thousand examples per classification (so one million total, as there are one thousand possible classifications), supervised learning starts catching up.

Video 13: Revisiting Our Goals

Alright, so at this point, we covered all topics we wanted to cover: Classification, Detection and Segmentation, 3-D Reconstruction, Image Generation, and Semi-supervised learning. Now, let’s revisit our original goals.

We wanted to understand which types of problems computer vision studies, which we have done through studying those five thrusts. We built an appreciation for how fast this field has progressed. We now know that in just a little over five years, we went from rather poor performance on image classification to reaching accuracies on benchmarks that start matching human-level performance. Similar for other benchmarks in segmentation, 3-D reconstruction, and so forth.

Now, naturally, these benchmarks are not fully general, and we are definitely not yet at human-level vision, but fast progress has been made, and current systems have a level of performance that can be useful for many applications. We now understand a lot of this progress was achieved through advances in deep learning. In turn, these advances were enabled by more compute, more data, and innovations in neural net architectures and data augmentation. We now have also seen how well current systems can perform and also where we can go find the best-performing systems.

Video 14: Introduction to Text Mining and Natural Language Processing

In this session, we want to introduce the landscape of problems in automated text analysis or text mining, while also reviewing basic intuitions and contemporary approaches. Much of the early work in text analysis stems from the information retrieval and data management communities; it evolved independently of the work in data mining and machine learning. However, the core ideas are fundamentally the same, so we will build on the same framework of unsupervised and supervised machine learning.

If we think in the most general of terms, text analysis is really about asking a question and getting an answer. As Bill Gates, the co-founder of the software giant Microsoft, notes, “Any ability to enhance this process enhances our ability to learn from others…”. In the simplest instance, we simply and directly talk to one another. However, this method of learning does not scale very well and is limited to the knowledge of the one person we ask.

Instead, we capture knowledge in documents, and so we enter the realm of text analysis. We can pose our question as a document search. The process of information retrieval searches for documents that are relevant to our question. The documents are relevant insofar as they may contain the answer to our question. Of course, we still have to read the documents in order to learn.

Information extraction embodies a number of different sub-tasks, among them named entity recognition and relationship Learning. Sometimes, the World Wide Web is simply the front-end or user interface to a database or catalog. A single web page might contain a table listing all of the courses in the course catalog or all of the products in a particular store. Using information extraction, we can extract and organize the details of course name, course number, meeting time and meeting location for each. More generally, any piece of text may contain details about people, locations, and objects. If we can extract entities and the relationships between them, we can often discover what we are looking for without needing to read the entire document.

At the limit, question answering identifies the relevant document and extracts the exact answer to the original question. For example, who is the CEO of Apple? In the year 2019, the CEO of Apple was Tim Cook. This evolution from information retrieval to information extraction to question answering uses the tools of Natural Language Processing or NLP.

NLU, or natural language understanding, is a subset of NLP that both enriches the ability of systems to parse documents or break sentences into their constituent, grammatical pieces. These are the technical steps in extracting information and answering questions.

NLG, or natural language generation, is the task of writing an answer to the question in prose. Of course, this entire narrative assumes that answers are written in text and stored in documents so that answers are found by searching, retrieving, and reviewing text. Of course, in this modern age, people rarely type document queries anymore. Instead, people use mobile devices to message. Chat bots are an alternative interface for learning through communication, and the development of chat bots builds on the same foundation.

Of course, rather than typing or texting, I might simply, today, say, ‘Siri’ or ‘Alexa’ or say, ‘Okay, Google.’ Automatic speech recognition takes spoken voice, converts my spoken query to text, and then proceeds as before. At the opposite end of the pipeline, Text-To-Speech technology converts my written answer back to an audio stream for me to hear. Given our limited time and the introductory nature of this segment, we will focus on the basics of information retrieval and the natural evolution to information extraction and then questioning-answering.

Video 15: Text Mining Fundamentals

The very basics of information retrieval are not unlike those of unsupervised learning. We start with a sample of data. In this case, every document in our collection constitutes an observation. We need to describe each observation in terms of a set of variables. The simplest such model is called the “Bag of Words” or “BOW”. It is a bag because we simply lump all of the words in the document together. The order of the words does not matter. This basic model is also referred to as the Vector Space Model (VSM). Every word is a variable, and the value of each variable is the presence, a one or absence, a zero of the word. As in the Bag of Words, order does not matter.

A simple query is likewise represented as a bag or a vector. The task of retrieval is simply the task of searching for those documents that are likely to contain the answer to the question. The intuition is that if a document contains all of the terms in my query, the documentation is probably relevant to my query, meaning that the answer is more likely than not to appear inside. So, in this simplest of cases, retrieval is simply a Boolean search. Documents are or are not relevant.

Of course, that simple model is also remarkably naïve. At the very least, documents may contain the same word more than once. Some documents are much longer and/or have many more words than others. Not all words are equally important. Documents and queries that share common words like “The” or “A” are somehow not as similar as documents and queries that share more semantically substantive words like “Seattle” or “Highlights.”

TF*IDF is an acronym used to normalize the vector representation in a simple way that attempts to address some of these basic concerns. The TF in TF*IDF refers to “Term Frequency”. When modeling a document, each variable representing a word is set to the number of times that word appears in the document. Words with higher frequency likely represent a larger share of the actual document meaning. But frequency alone is a problem because words are not all equally substantive.

The IDF in TF*IDF refers to “Inverse Document Frequency”. For a given word, Document Frequency refers to the number of documents in the collection which contain the given word. If document frequency is high, the word likely has little discrimination value. It does not help us distinguish between relevant documents and irrelevant documents. So, we multiply term frequency by the inverse document frequency in order to get a normalized vector representing each document.

To match documents to a query, we compute the similarity of each document vector to the query vector. Comparing vectors to one another should seem familiar. We perform this very operation in unsupervised learning applications such as *k*-means clustering. Using the Manhattan Distance or Euclidean distance, we can compute how similar (or close) a document is to the query. Documents that are more relevant to the query are presumably closer to or more similar to the query.

Just for completeness, you should know that basic models of information retrieval interpret documents as multi-dimensional vectors in space. The most common comparison between two vectors in space is the measure of their angular distance. Simply put, the cosine of the angle between two vectors is computed as the dot product between the two vectors. It is common to hear people speak of the cosine distance, and this is what people are referring to.

Video 16: Information Retrieval

One way of thinking about information retrieval is ranking all documents in their order of relevance. Documents closer to the query are more relevant. However, one can also think of information retrieval in a binary sense. By setting a threshold on distance, documents whose distance to the query satisfies the threshold are relevant. All other documents are irrelevant.

By setting a threshold, we set up information retrieval as a classification problem. We predict that relevant documents contain the answer to the query. We predict that irrelevant documents do not contain the answer to the query. From this perspective, we can see why information retrieval, information extraction, question answering, and other common text analytics tasks are typically evaluated using the same measures as those in classification.

Of course, this does mean that we require labeled data. To benchmark an information retrieval algorithm, someone has to manually review a document collection and determine “yes” or “no” – whether a particular document is relevant to a particular query. By moving the threshold up and down in terms of relevance or distance, we adjust which documents are predicted as relevant to a particular query. In the field of information retrieval and text analytics, more generally, the common metrics for summarizing the confusion matrix are precision and recall. In addition, text analytics researchers often report the F1 measure, which aggregates precision and recall into a single metric defined as the harmonic mean with a parameter of 0.5.

Counting words in order to model documents and search for relevant documents is an extremely simple idea. One significant challenge is the problem of synonyms. People often use different words to refer to the same concept. For example, is a seismic wave and a sound wave the same thing? Based only upon word counts, how can the computer determine that two words have the same meaning? Without going into the mathematical details, the intuition behind latent semantic analysis captures the idea that words that co-occur with common words are perhaps likely to have the same meaning.

To capture this idea, we can take all of the document vectors and line them up to create what we call a “word document matrix.” Every row represents one document vector. Every column represents the TF*IDFy, adjusted term frequency counts for a particular word across all documents in the collection. Applying a mathematical computation, this matrix and all of the words in it are separable into a set of constituent matrices which have fewer columns or rows. Every column or word in the original word document matrix is mapped into a smaller space of topics. By construction, there are fewer topics than there are words because every topic captures the idea that multiple words may represent the same topic. Essentially, these are synonyms of one another. Using the same cosine measure of distance, we can compute the relevance of a document in the topic space to ensure that we do not miss documents that use synonyms.

Video 17: Word Embeddings

More recently, a new method for representing the co-occurrence relationship of words has emerged. LSA takes the large dimensional space of words and shrinks it down into a set of latent meta-concepts or topics to represent the idea that different combinations of words can express the same thing. Word embeddings have the same effect of modeling semantically related terms. As the name “word embeddings” suggests, a word in a document is characterized not by its frequency but by its word context.

Each word is represented by a row or vector. The variables describing the focal word are all words in the vocabulary, along with how far apart they are – their “context” to one another. For the focal word “Seattle” in Document 1, the word “Seahawks” is one word after – so the variable (Seahawks plus one) has the value “1”. The word “jerseys” is two words after “Seattle”, so the variable (jerseys plus two) has the value “1”. There is also a variable for the appearance of the word “jerseys plus three”, which has the value of “0” for Document 1. You could imagine a different sentence, such as “seattle Seahawks football jerseys.” For this new sentence, the value of “jerseys plus two” is “0”, and “jerseys plus three” is “1”. Word embeddings can represent words in a single document.

The real power of embeddings, however, comes from modeling individual words in an entire document collection. Consider our original four documents. If we combine the embeddings for the word “Seattle” from all four documents, we end up with a vector that looks like this. Other words in our embeddings look like this. Of course, you can see that we omitted a number of different variables because they are not particularly relevant to this collection. The variable “jerseys plus three” is not represented because it is “0” everywhere. From the word embeddings, it is now easy to see how “Seattle” and “Denver” are semantically similar to one another. They are both cities. Likewise, “Seahawks” and “Broncos” are semantically similar. They are both sports teams. This is all from context. Given a large enough collection of documents, the word documents can begin to represent an entire language.

Word2vec is a collection of word embeddings open-sourced by Google. It contains embeddings for billions of English words and their respective contexts. An analysis of Word2vec reveals the semantic relatedness of words in English. Word2vec captures the concept of gender not only in specific nouns but also in specific roles: man and woman, as well as king and queen. Word2Vec captures the relatedness between past and present tense. “Walked” versus “walking”. It expresses geographic relationships, such as a country and its capital city. All of this from word contexts in a sufficiently large corpus.

Armed with a dictionary of word embeddings, we change our model of document search. Rather than modeling a document in terms of word frequencies, we can model a document in terms of its word contexts. Each word in the document is represented by a word-embedding vector. Rather than a descriptive analytics framework, where document vectors describe a single document in a collection, retrieval now looks more like a supervised learning exercise, where the training data constitutes the universe of documents used to configure or train the word embeddings.

To compare a document to a query, we now compute the similarity of their respective word embeddings. Note the contrast to our original strategy for computing synonyms. The LSA matrix factorization approach requires that our document collection contains all the necessary word contexts. With Word2vec, we can incorporate semantic relationships from a larger universe of documents. We can learn from other domains or other document collections.

Of course, this described approach is still limited in the same way that any traditional, supervised learning algorithm is limited. If the semantic relationships are not embedded within the larger universe of documents encoded in the word embeddings, our document retrieval cannot account for it. In the same way, edge cases that do not appear in any supervised learning training set are not captured in a traditional supervised learning model. This is also important because it highlights the intuitive distinction between general-purpose knowledge and domain-specific knowledge. If you employ Word2vec in your applications, you are employing a dictionary of word embeddings generated from the universe of Google search documents. That knowledge may inaccurately reflect the semantics of highly specialized vocabularies, such as medicine or finance. We will say more about how to adjust for this a bit later today.

Video 18: Question Answering as Prediction

Having found the interesting documents, our next challenge is to extract the relevant details rather than needing to read everything. There is a broad task called document summarization that we will skip past for reasons of time. Instead, we jump directly into the problem of information extraction. We can separate out the challenge of extracting data from text into two categories: Semi-structured data is text documentation that loosely follows a template. A LinkedIn profile, a Facebook page, a list of Amazon products, or a list of Yelp restaurant reviews are all examples of semi-structured data. These are text documents that have a template underneath. We can exploit the regularity in the template to extract the details within the text. Consider the example here of a classified advertisement declaring to housing rental opportunities.

Without delving into the details, we can say that there is a way to specify the pattern in a rental classified ad. In this instance, we know that if we see the character string ‘bd’ preceded by a number, that number likely refers to the number of bedrooms in the rental. Likewise, if we see the symbol “dollar sign” followed by a number, chances are that number is the price of the rental. Using only this pattern, we can see that this classified advertisement actually includes two distinct rental opportunities, one for a one-bedroom and one for a three-bedroom. You can hopefully see how we can use supervised machine-learning techniques to learn these information extraction patterns.

Given a collection of classified ads and given a set of labels that pairs each classified ad with their respective number of bedrooms and rental prices, we can learn a pattern and report the precision and recall with which our pattern successfully extracts the correct number of bedrooms and price.

Of course, not all interesting information is contained in semi-structured documents. What do we do when there is no apparent underlying template to the data? What we call "unstructured" data. State-of-the-art work on extracting information from a collection of text treats the problem like a prediction in supervised machine learning. More particularly, we can think of question answering as predicting an answer to a given question. For example, given the words "what is the highest mountain in the world," a good prediction might result in the words "Mount Everest."

To frame the task of question answering as a prediction, let us start with a simple example. Many people who read science fiction are familiar with the author Isaac Asimov. In his story collection I, Robot, Asimov first introduces his Laws of Robotics. The Zeroth Law reads, "A robot may not harm humanity, or by inaction, allow humanity to come to harm." Now, suppose that we see the word sequence "The Zeroth Law: A robot may not." The input variables represent the question. The outcome variable, the label that we want to predict, is the next word in the sequence. If you were asked to predict the next word in the sequence, based upon our knowledge as represented by "The Zeroth Law", chances are that you would guess the next word as "harm." What if you appended the word "harm" to the original sequence? Now, what would you guess is the next word? "humanity". Add humanity to the sequence. Now, what would you guess? "or." This guessing game seems pretty easy at first.

Let us make this a bit more complicated. What if we add another sentence into our knowledge base? For instance, what if we add the First Law of Robotics? Now, what if we ask a question described by the following input sequence of words, "A robot may not". The answer that we wish to predict is the next word in the sequence. What word do you think comes next? Based upon our sample, there is a 50% chance that the next word is "injure" and a 50% chance that the next word is "harm." This, then, is the intuition behind generative models like large language models.

From the data in our knowledge base, we learn probability distributions to help predict what comes next. Of course, rather than just two sentences, LLMs are armed with thousands of sentences or sequences of words. News stories, customer reviews, encyclopedia entries, works of creative fiction, and virtually any other text available online provide the knowledge or examples from which an LLM learns what the next word is. Given all of the publicly accessible Web pages out there, if we type the question "what is the highest mountain in the world," the most likely next word in the sequence is "mount". And if we add "mount" to our question (which now reads "what is the highest mountain in the world, mount?"), it is highly likely that the word with the highest probability of appearing next is "Everest".

Video 19: LLMs and self-supervised learning

Large Language Models, or LLMs, are an example of Generative Models. To understand how an LLM works, we return to the supervised machine learning framework of prediction. In supervised machine learning, recall that we collect a sample of observations, such as pictures of fire trucks. For each observation in the sample, we select a number of input variables that describe an observation. For example, the number of axles or the color of the vehicle. For each observation, we also have an outcome variable, such as the binary class we wish to predict. In the case of vehicles, we want to predict "yes, it is a firetruck" or "no, it is not a firetruck."

Supervised learning requires data from which to learn. We need to know the correct label for each observation in our sample. In other words, an external source, like a human expert, assigns the correct label to each observation that we use for training or teaching our prediction model. The goal of supervised learning is to use the labeled examples to learn a relationship between the input variables and the outcome variable. Of course, every input variable and outcome variable has a distinct data type or representation. The color of a vehicle might be represented by wavelength in nanometers, frequency in Hertz, or as a string of characters. The output variable might be a boolean True for firetruck or the number one in binary. The nature of the variable representations impacts the nature of the learned relationship. For example, we might fit a linear function that relates the inputs to the outputs, or, we might learn a decision tree of categorical rules that iteratively splits or partitions labeled observations into ever smaller subsets.

Because an LLM knows what the next word is, you might think of this as supervised learning. Every sentence provides evidence of which words follow other words. When we see “The Zeroth Law: A robot may not,” we know that the next word is “harm”. On the other hand, “next word” is not supplied by an external third party or expert. Unlike pictures of firetrucks, the prediction, the “label”, or the “next word” is contained within the data itself. For this reason, you might think of this as unsupervised learning. Because models like this fall somewhere between unsupervised and supervised, they are often referred to as self-supervised. Each sentence contains within itself the “label” of what word appears next.

Continuing with the framework of supervised learning, the sample is a set of documents. The input variables and the outcome variable are words in the sample. To represent each word, instead of using something like “Term Frequency”, we use the “Embeddings” that we described earlier. For example, from the Zeroth Law of Robotics, we can represent each word by its proximity to every other word. One way of thinking about embeddings is that they reflect a relative, spatial relationship between words. How far apart one word is from every other word? LLMs use another spatial relationship as well. Think of the position encoding as an absolute measure. Where in the document or sentence does a word appear? Taken together, embeddings and word position encodings represent the variables in our LLM. “Self Attention” describes the concept of local context. As the LLM processes a string of text, how far forward and how far backward in the string does the machine read when defining the context of a word? For example, the second law of robotics reads, “A robot shall obey any instruction given to it by a human.” To what does the pronoun “it” refer? Depending upon how far forwards and backwards we read, possible nouns include “robot,” “instruction” and “human.”

To predict the next word, LLMs like ChatGPT use what is called a transformer architecture. The transformer architecture actually has two parts: An “encoder” and a “decoder.” The “transformer” is so-called because, like an electrical transformer, an input signal is transformed from one representation to another. The electrical transformer steps down voltage from high-tension, like the power lines outside your building, to internal household current. The Machine Learning transformer transforms text like a written paragraph or a question from one representation to another. On the Encoder side, the original text or query is first transformed into a unique representation that combines an embedding and a position encoding.

The query is then passed into a multi-layer neural network in the same spirit as the neural networks introduced by Zsolt. Each subsequent transformation further reduces the original text into a deep representation that represents the underlying meaning. From the deep representation, the decoder side of the transformer steps the text up through another series of neural network layers. In the task of language translation, the original input text is transformed from one language to another. In question answering, the original text is transformed by predicting and appending the next word in the sequence.

Video 20: Generative models: LLMs and Beyond

Importantly, the transformer architecture applies to more than just text. For example, consider propylene glycol. This is a widely used chemical additive that appears in medicine, cosmetics, and food products. Yet, it was named “Contact Allergen of the Year in 2018.” Instead of releasing this additive into the public, what if chemists could have predicted the negative side effects based upon its structural properties? For example, if you think back to your organic chemistry class, propylene glycol looks like this. It has a two-dimensional shape like this and a three-dimensional shape like this.

Borrowing from our supervised learning framework, if we treat the sample as the set of all organic molecules and the input variables as the vocabulary of atomic elements, we can think of representing the atomic elements using embeddings to reflect their position relative to one another. The position encoding reflects the absolute position of each element, and self-attention provides a similar window for looking forwards or backwards through atomic structure for binding sites.

The transformer architecture might then transform the textual representation of an organic molecule into a prediction of its allergenic properties and other molecular characteristics.

For example, a huge challenge in medicine today is the emergence of superbugs or bacteria that are resistant to our most powerful antibiotics. In the summer of 2023, scientists at MIT and McMaster University described an experiment to find new treatments for superbugs. The scientists started with the molecular structure of 7,500 different chemical compounds with known properties for inhibiting antibiotic-resistant bacterial growth. From this starting point, the team constructed a graph neural network or GNN, a more general type of transformer architecture, to capture and represent spatial relationships. Using their GNN, the researchers computationally constructed and screened more than 6,000 possible new antibiotics. Although there is no substitute for actually synthesizing and testing antibiotics, this computational approach to drug discovery greatly accelerates the timeline and allows for more focused, cost-effective laboratory synthesis and testing.

Large Language Models are just one application of the neural networks that underlie generative models. Whether applied to text or molecular structure or, image processing, or audio signals, the bottom line is that these models require a tremendous amount of data, and the required data is growing exponentially. In practice, most organizations have no hope of building such models entirely from scratch on their own. The challenge is, therefore, to explore techniques for building on top of someone else’s investment in data and model building.

Some of the earliest work in reusing multi-layered neural networks is a technique called transfer learning. Transfer learning is so called because researchers begin with a model trained in one domain and then “transfer” that model to a different domain. This picture depicts a convolutional neural network of the kind described by Zsolt for classifying the digital images of objects like dogs, cats, planes and cars. The original image classifier is trained on a large collection of images.

Suppose now that you would like to train an image classifier to process images to detect product defects in a manufacturing process. The new problem is an entirely different application. Rather than designing a new classifier entirely from scratch, suppose that you started with the original image classifier. The goal is to reuse a large number of layers from the original neural network. Then, with comparatively few images from the new domain, how might you add or otherwise substitute new layers to the original network?

Two important points. First, one or more dense layers or convolutional layers from the original model are re-engineered. Second, the re-engineering is based upon a smaller data set. Indeed, the term “one-shot” learning is sometimes used in this context and implies that as little as one image is enough to adapt an existing model to a new domain.

Transfer learning is but one way of reusing an existing model. LLMs highlighted a second way of reusing an existing model. This second technique is often called prompt engineering. In tasks like language translation or question answering, the input to the encoder is a passage of text or a question. More generally, the input is sometimes called a “prompt.” Instead of simply transcribing a prompt, the user might add additional context to guide or otherwise constrain the nature of the output. For example, if we wanted our LLM to solve a math puzzle, we could give an example problem and an example answer as part of the initial prompt. By providing an example of a model problem and solution, we are guiding the output prediction to give a similar solution.

In summary, we reviewed both the history and the state-of-the-art in text analysis. We did so by first framing our discussion in the context of the very human problem of learning. We can learn from asking someone else a question, or we can learn by searching for documents and reading those documents to learn ourselves. Advances in text analytics hold promise for transforming the whole of our written documentation into a database for real-time, automated question-answering. We can ask a question and receive a narrative response that directly answers our question. The knowledge behind that answer lies in the collection of documents from which the computer learns. Progress in this space began in the field of information retrieval using a simple vector space representation. The frequency-based descriptive approach to text analysis gave way to the predictive approach exemplified by large language models which exploit neural networks.