AI in math and physics

Michael R. Douglas
Stony Brook University

Abstract

Some applications of machine learning and AI in physics and math:
- ML in physics simulations
- ML and the string landscape
- Mathematical search engines
- AI for mathematical theorem verification
Brief review of deep learning

What is deep learning? A set of computational advances which solve many machine learning problems using the same general methods:

- Regular models (minimal prior structure, e.g. MLP = multilayer perceptron, CNN = convolutional network).
- Overparameterized (yet doesn’t overfit – “blessing of dimension”).
- Differentiable objective functions which lead to tractable optimization problems (more convex than we thought).

Discovered by study of examples, and general principles and scope remain unclear.

Some prototypical tasks, datasets and systems:

- Supervised learning: image recognition (MNIST, CIFAR-10, Imagenet) mostly by CNNs (LeNet, Alexnet, Inception, ResNet)
- Reinforcement learning: game playing (AlphaGo, AlphaZero, ...)
- Unsupervised learning: language models (Bert, GPT-2, ...)
MLP or fully connected feedforward network:

\[ f(x) = \sigma \left( \sum_i W^{(L)}_i \sigma \left( \sum_j W^{(L-1)}_{i,j} \sigma \left( \ldots \sigma \left( \sum_j W^{(1)}_{m,n} x^n \right) \right) \right) \right) \]

\( \sigma(x) \) is typically \((1 + \text{sgn} x)/2\) (ReLU) or \(\tanh x\).

- CNN is similar where \(W_{i+\alpha,j+\alpha} = W_{i,j}\).
- ResNet: use \(\delta_{i,j} + W_{i,j}\) for the weights.
- GCN: start with embedding features of node \(i\) into vector \(\vec{h}^{(1)}_i\); then the \(k\)'th layer aggregates vectors for neighboring vertices and applies a nonlinear layer

\[ \vec{h}^{(k+1)}_i = \sigma \left( W^{(k)} \cdot \text{Combine} \left( \vec{h}^{(k)}_i; \bigotimes_{j \in \mathcal{N}(i)} h^{(k)}_j \right) \right) \]
Some theoretical questions about deep learning (Oct 2019, IAS and DeepMath conferences):

- How can models with $N_{\text{parameters}} \gg N_{\text{data}}$ generalize?
  - Is $S_{\text{parameters}} < N_{\text{data}}$, and if so can we compress models?
  - Is this due to some implicit regularization?
  - Dependence on optimization technique?

- How does test error $E$ depend on $N_{\text{data}}$?
  - Many works suggest $E \sim N^{-\alpha}$ with $0 < \alpha \leq 0.5$.
  - Surely often the case, but in general seems in tension with complexity theory, which might suggest $E \sim -\log N$.

- Are there more tractable limits? For example, infinite width $\rightarrow$ neural tangent kernel (Jacot et al 2018).

Some broader topics receiving widespread attention:

- How do models learn and represent latent structure in dataset?
  - For example, do language models learn grammar?
- Integration of differentiable and symbolic computation
- Search for new architectures, automated selection
Some theoretical questions about deep learning (Oct 2019, IAS and DeepMath conferences):

- How can models with $N_{\text{parameters}} \gg N_{\text{data}}$ generalize? Is $S_{\text{parameters}} < N_{\text{data}}$, and if so can we compress models? Is this due to some implicit regularization? Dependence on optimization technique?

- How does test error $E$ depend on $N_{\text{data}}$? Many works suggest $E \sim N^{-\alpha}$ with $0 < \alpha \leq 0.5$. Surely often the case, but in general seems in tension with complexity theory, which might suggest $E \sim -\log N$.

- Are there more tractable limits? For example, infinite width $\rightarrow$ neural tangent kernel (Jacot et al 2018).

Some broader topics receiving widespread attention:

- How do models learn and represent latent structure in dataset? For example, do language models learn grammar?

- Integration of differentiable and symbolic computation

- Search for new architectures, automated selection
Some theoretical questions about deep learning (Oct 2019, IAS and DeepMath conferences):

- How can models with $N_{\text{parameters}} \gg N_{\text{data}}$ generalize?
  Is $S_{\text{parameters}} < N_{\text{data}}$, and if so can we compress models?
  Is this due to some implicit regularization?
  Dependence on optimization technique?

- How does test error $E$ depend on $N_{\text{data}}$?
  Many works suggest $E \sim N^{-\alpha}$ with $0 < \alpha \leq 0.5$.
  Surely often the case, but in general seems in tension with complexity theory, which might suggest $E \sim -\log N$.

- Are there more tractable limits? For example, infinite width $\rightarrow$ neural tangent kernel (Jacot et al 2018).

Some broader topics receiving widespread attention:

- How do models learn and represent latent structure in dataset?
  For example, do language models learn grammar?

- Integration of differentiable and symbolic computation

- Search for new architectures, automated selection
Some theoretical questions about deep learning (Oct 2019, IAS and DeepMath conferences):

- How can models with $N_{\text{parameters}} \gg N_{\text{data}}$ generalize?
  
  Is $S_{\text{parameters}} < N_{\text{data}}$, and if so can we compress models?
  
  Is this due to some implicit regularization?
  
  Dependence on optimization technique?

- How does test error $E$ depend on $N_{\text{data}}$?
  
  Many works suggest $E \sim N^{-\alpha}$ with $0 < \alpha \leq 0.5$.
  
  Surely often the case, but in general seems in tension with complexity theory, which might suggest $E \sim -\log N$.

- Are there more tractable limits? For example, infinite width $\rightarrow$ neural tangent kernel (Jacot et al 2018).

Some broader topics receiving widespread attention:

- How do models learn and represent latent structure in dataset?
  
  For example, do language models learn grammar?

- Integration of differentiable and symbolic computation

- Search for new architectures, automated selection
New neural architectures

Fully connected networks are universal but not necessarily effective. Many datasets have structure which motivates a more restricted model. The prime example is translation invariance of images, which is encoded in CNNs.

A major direction of progress is to find new architectures adapted to other tasks and datasets:

- Graphical and tree-structured data (parse trees, knowledge graphs, physical science examples): Laplacian eigenmaps and embedding techniques, GNNs (graph NNs), GCNs (graph convolution networks) and other message passing techniques, ...

- Languages (sequences with latent long-range structure): neural parsers, attention, transformer architecture.

- Tasks using large databases: sparse attention, memory layers

- Many more: neural Turing machines, neural state machine, graph attention networks, etc..
New neural architectures

Fully connected networks are universal but not necessarily effective. Many datasets have structure which motivates a more restricted model. The prime example is translation invariance of images, which is encoded in CNNs.

A major direction of progress is to find new architectures adapted to other tasks and datasets:

- Graphical and tree-structured data (parse trees, knowledge graphs, physical science examples): Laplacian eigenmaps and embedding techniques, GNNs (graph NNs), GCNs (graph convolution networks) and other message passing techniques, ...
- Languages (sequences with latent long-range structure): neural parsers, attention, transformer architecture.
- Tasks using large databases: sparse attention, memory layers
- Many more: neural Turing machines, neural state machine, graph attention networks, etc.
From the NSF AI and physics solicitation:

*NSF encourages AI Research Institutes in Physics that advance AI and accelerate discovery in the physical sciences. Examples of this might include: (i) improving and optimizing operations, real-time event selection, classification, feature extraction, reconstruction, and analysis at dataflow-intensive facilities; (ii) accelerating multi-scale, multi-physics simulations for multi-messenger astrophysics, quantum chromodynamics, cosmology, and plasma physics; (iii) exploring the very large space of potentially viable string theories ("string landscape"); (iv) developing and validating predictive dynamical models of complex, far-from-equilibrium systems; (v) improving the understanding of the physics principles behind genome packing and the resulting genome architecture and dynamics; (vi) and co-developing improved physical models of brain function and new AI architectures.*
The physical sciences are characterized by well established microscopic models:

- **Classical physics**: particle mechanics, Maxwell’s equations and other field theories, Navier-Stokes, MHD.
- **Quantum field theory**: the Standard Model of particle physics, lattice gauge theory.
- **Many body quantum physics**: quantum chemistry, lattice systems, density functional models.
- **Phenomenological models**: molecular dynamics and protein folding, reaction networks.
- **Mathematical and other idealized models**: supersymmetric and topological field theory, random matrix models, spin glasses, models of pattern formation and growth.
Most of these models have well developed numerical approximations and computer simulation is a much used tool. Simulation has led to many important discoveries and conceptual advances. For example, the theory of chaos in classical mechanics began with numerical simulations.

Many models are probabilistic, because of thermal effects and noise, quantum indeterminacy, uncertain initial conditions and classical chaos, \textit{etc}.. Statistics will thus enter in analyzing the results. Let us skip over this established topic and move on to more recent applications of ML. These include

- Learning models from data.
- Making better and/or more efficient approximations of the state, using function approximations such as CNNs or tensor networks.
- Replacing parts of the simulation with learned approximations. For example, replace the subgrid dynamics in a climate simulation or an astrophysical simulation, with a learned model of this dynamics.
Let us consider the classical mechanics of $N$ particles in 3 dimensions. Suppose we measure their positions at a sequence of times $t_1, t_2, \ldots$, to get a time series $\vec{x}_i(t_n)$. Can we automatically infer the equations of motion, or equivalent information such as a Hamiltonian?

Many versions of this have been tried, some assisting humans (Abelson and Sussman 1989) and some automatic (Langley 1979, Dzeroski and Todorovski 1993; Schmidt and Lipson 2009). Most of these were symbolic, searching through candidate equations to find models. For example, Schmidt and Lipson used genetic programming.

Recently many groups are trying deep learning methods, e.g. Battaglia et al NeurIPS 2016, Chang et al ICLR 2017, Chen et al 1806.07366, Greydanus et al 1906.01563, Sanchez-Gonzalez et al 1806.01242, 1909.12790, etc.
Learning models from data

Let us consider the classical mechanics of $N$ particles in 3 dimensions. Suppose we measure their positions at a sequence of times $t_1, t_2, \ldots$, to get a time series $\vec{x}_i(t_n)$. Can we automatically infer the equations of motion, or equivalent information such as a Hamiltonian?

Many versions of this have been tried, some assisting humans (Abelson and Sussman 1989) and some automatic (Langley 1979, Dzeroski and Todorovski 1993; Schmidt and Lipson 2009). Most of these were symbolic, searching through candidate equations to find models. For example, Schmidt and Lipson used genetic programming.

Recently many groups are trying deep learning methods, e.g. Battaglia et al NeurIPS 2016, Chang et al ICLR 2017, Chen et al 1806.07366, Greydanus et al 1906.01563, Sanchez-Gonzalez et al 1806.01242, 1909.12790, etc.
Let us consider the classical mechanics of $N$ particles in 3 dimensions. Suppose we measure their positions at a sequence of times $t_1, t_2, \ldots$, to get a time series $\vec{x}_i(t_n)$. Can we automatically infer the equations of motion, or equivalent information such as a Hamiltonian?

Many versions of this have been tried, some assisting humans (Abelson and Sussman 1989) and some automatic (Langley 1979, Dzeroski and Todorovski 1993; Schmidt and Lipson 2009). Most of these were symbolic, searching through candidate equations to find models. For example, Schmidt and Lipson used genetic programming.

Recently many groups are trying deep learning methods, e.g. Battaglia et al NeurIPS 2016, Chang et al ICLR 2017, Chen et al 1806.07366, Greydanus et al 1906.01563, Sanchez-Gonzalez et al 1806.01242, 1909.12790, etc.
A straightforward approach tried by many groups is to use an NN to parameterize the vector field $\vec{f}$ which defines the ODE,

$$\frac{d\vec{z}}{dt} = \vec{f}(\vec{z}(t), t); \quad \vec{f} = \text{NN}(W, \vec{z}(t)).$$

The loss $L$ can be the squared error for the solution $\vec{z}(t_i)$ to predict the observations $\vec{x}(t_i)$.

The parameterized ODE can also be thought of as a continuous version of a ResNet (or “neural ODE”), with the layer index replaced by $t$. This suggests new ways to compute $dL/dW$, for example the adjoint sensitivity method as used by Chen et al. The idea is to solve a time-reversed ODE for $\vec{a}(t) = \partial L/\partial \vec{z}(t),$

$$\frac{d\vec{a}}{dt} = \vec{a}(t) \cdot \left( \frac{\partial \vec{f}(W, \vec{z}(t), t)}{\partial \vec{z}(t)} \right),$$

with the initial conditions $\vec{a}(t_f) = \partial L/\partial \vec{z}(t_f)$. This is a continuous version of backpropagation. Combining this with $\partial f/\partial W$ we get $dL/dW$. 

Michael R. Douglas (Simons Center)
A straightforward approach tried by many groups is to use an NN to parameterize the vector field \( \vec{f} \) which defines the ODE,

\[
\frac{d\vec{z}}{dt} = \vec{f}(\vec{z}(t), t); \quad \vec{f} = \text{NN} (W, \vec{z}(t)).
\]

The loss \( L \) can be the squared error for the solution \( \vec{z}(t_i) \) to predict the observations \( \vec{x}(t_i) \).

The parameterized ODE can also be thought of as a continuous version of a ResNet (or “neural ODE”), with the layer index replaced by \( t \). This suggests new ways to compute \( dL/dW \), for example the adjoint sensitivity method as used by Chen et al. The idea is to solve a time-reversed ODE for \( \vec{a}(t) = \partial L / \partial \vec{z}(t) \),

\[
\frac{d\vec{a}}{dt} = \vec{a}(t) \cdot \left( \frac{\partial \vec{f}(W, \vec{z}(t), t)}{\partial \vec{z}(t)} \right),
\]

with the initial conditions \( \vec{a}(t_f) = \partial L / \partial \vec{z}(t_f) \). This is a continuous version of backpropagation. Combining this with \( \partial f / \partial W \) we get \( dL/dW \).
Many variations on this:

- Replace $\dot{z} = f(z)$ by Hamiltonian dynamics, $\dot{p} = \frac{\partial H}{\partial q}$ and $\dot{q} = -\frac{\partial H}{\partial p}$, and represent $H$ by an NN.

- Use prior knowledge about the model, for example bodies $i$ and $j$ are not allowed to interact. The function approximator can be a GNN.

Results from Sanchez-Gonzalez et al 1909.12790:
Better state approximations

Many physical models have a large or even infinite number of degrees of freedom.

- Maxwell’s equations postulate an electromagnetic field \((\vec{E}, \vec{B})\) whose components are continuous functions on space-time.
- The equations governing media (Navier-Stokes, theory of elasticity, MHD) idealize the material (made up of atoms) in terms of continuous density or displacement functions.
- The wave function in Schrödinger’s equation for \(N\) particles is a complex function on \(\mathbb{R}^{3N}\).

To simulate them, one must approximate these functions by a finite dimensional parameterized family of functions.

Standard numerical methods use very simple (usually linear) parameterizations. For PDE’s one usually uses local discretizations, finite elements, or FFT. For QM one expands the wave function in a “nice” basis, for example molecular orbitals for quantum chemistry.
Let’s look at using NN’s in “metric/topology” optimization problems:

\[ \mathcal{L} = \int_M \frac{1}{2} G (\partial \phi)^2 + J \phi \]

\[ + \lambda \left( V - \int G \right) + (G \text{ well-formed}) \]

- M is a manifold or graph whose topology need not be fixed.
- G is a Riemannian metric and/or density.
- J is a source or boundary condition.
- \( \phi \), the response, can be determined by solving \( \Delta_G \phi = J \).
- There may also be a statistical or quantum energy \( \log \det \Delta_G \).

Goal: optimize G. Ubiquitous, ranging from mechanical engineering (M a domain in 2 or 3 dimensions, \( \phi \) strain vector) to string theory (M a six-dimensional Calabi-Yau, \( \phi \) quantized three-form flux).
In topology optimization, $M$ is a fixed region in $\mathbb{R}^2$ or $\mathbb{R}^3$, $J$ is an applied force, and $G$ is the density of material. The structure and nonlinearity in the problem comes from the constraints: $0 \leq G \leq 1$ and $V = \int G$. Usually $G$ is represented in position space (pixels). There are many almost flat directions in the objective function and ad hoc devices are used to regularize.

NN’s were first tried for structural optimization long ago, e.g. see Papadrakakis and Lagaros, 2002. We quote the recent work of Hoyer, Sohl-Dickstein and Greydanus, arXiv:1909.04240. They represent $G$ in terms of the output of a CNN, whose inputs are also trainable weights. Its architecture is the U-net of Ulyanov et al “Deep image prior” 2018, which starts with a dense layer and then five 2D convolutional layers. The outputs $z(x)$ are then mapped to $G(x)$ satisfying the constraints as $G(x) = 1/(1 + \exp(z(x) - b))$.

Let’s compare the results with those from standard methods for design of a 2D multi-story building, built to support forces applied on a series of horizontal lines (floors).
In topology optimization, $M$ is a fixed region in $\mathbb{R}^2$ or $\mathbb{R}^3$, $J$ is an applied force, and $G$ is the density of material. The structure and nonlinearity in the problem comes from the constraints: $0 \leq G \leq 1$ and $V = \int G$. Usually $G$ is represented in position space (pixels). There are many almost flat directions in the objective function and ad hoc devices are used to regularize.

NN’s were first tried for structural optimization long ago, e.g. see Papadrakakis and Lagaros, 2002. We quote the recent work of Hoyer, Sohl-Dickstein and Greydanus, arXiv:1909.04240. They represent $G$ in terms of the output of a CNN, whose inputs are also trainable weights. Its architecture is the U-net of Ulyanov et al “Deep image prior” 2018, which starts with a dense layer and then five 2D convolutional layers. The outputs $z(x)$ are then mapped to $G(x)$ satisfying the constraints as $G(x) = 1/(1 + \exp(z(x) - b))$.

Let’s compare the results with those from standard methods for design of a 2D multi-story building, build to support forces applied on a series of horizontal lines (floors).
In topology optimization, $M$ is a fixed region in $\mathbb{R}^2$ or $\mathbb{R}^3$, $J$ is an applied force, and $G$ is the density of material. The structure and nonlinearity in the problem comes from the constraints: $0 \leq G \leq 1$ and $V = \int G$. Usually $G$ is represented in position space (pixels). There are many almost flat directions in the objective function and ad hoc devices are used to regularize.

NN’s were first tried for structural optimization long ago, e.g. see Papadrakakis and Lagaros, 2002. We quote the recent work of Hoyer, Sohl-Dickstein and Greydanus, arXiv:1909.04240. They represent $G$ in terms of the output of a CNN, whose inputs are also trainable weights. Its architecture is the U-net of Ulyanov et al “Deep image prior” 2018, which starts with a dense layer and then five 2D convolutional layers. The outputs $z(x)$ are then mapped to $G(x)$ satisfying the constraints as $G(x) = 1/(1 + \exp(z(x) - b))$.

Let’s compare the results with those from standard methods for design of a 2D multi-story building, build to support forces applied on a series of horizontal lines (floors).
Neural (L-BFGS), pixel (MMA), pixel (L-BFGS) at $t = 0$ (Hoyer et al)
Neural (L-BFGS), pixel (MMA), pixel (L-BFGS) at $t = 10$ (Hoyer et al)
Neural (L-BFGS), pixel (MMA), pixel (L-BFGS) at $t = 20$ (Hoyer et al)
Neural (L-BFGS), pixel (MMA), pixel (L-BFGS) at $t = 30$ (Hoyer et al)
Neural (L-BFGS), pixel (MMA), pixel (L-BFGS) at $t = 40$ (Hoyer et al)
Neural (L-BFGS), pixel (MMA), pixel (L-BFGS) at $t = 90$ (Hoyer et al)
Hoyer et al tried this on 116 problems and report better results than the standard MMA approach. As in the example, the CNN produces qualitatively different and arguably simpler designs. Clearly there is something to understand here.

And, a good part of what we need to understand is the same as for an image recognition task: function approximation classes, priors, local minima and overparameterization, dependence on optimization methods, etc. But a less data-driven task lets the CNN+optimization “show us its priors.”

We could also turn this into a generative model for images, perhaps by randomizing the choice of support surfaces. My point is not that it is so different from generative models commonly used in statistics. In fact I want to make the opposite point, that these problems share much with learning problems and should be studied together.
Hoyer *et al* tried this on 116 problems and report better results than the standard MMA approach. As in the example, the CNN produces qualitatively different and arguably simpler designs. Clearly there is something to understand here.

And, a good part of what we need to understand is the same as for an image recognition task: function approximation classes, priors, local minima and overparameterization, dependence on optimization methods, *etc.*. But a less data-driven task lets the CNN+optimization “show us its priors.”

We could also turn this into a generative model for images, perhaps by randomizing the choice of support surfaces. My point is not that it is so different from generative models commonly used in statistics. In fact I want to make the opposite point, that these problems share much with learning problems and should be studied together.
Hoyer et al tried this on 116 problems and report better results than the standard MMA approach. As in the example, the CNN produces qualitatively different and arguably simpler designs. Clearly there is something to understand here.

And, a good part of what we need to understand is the same as for an image recognition task: function approximation classes, priors, local minima and overparameterization, dependence on optimization methods, etc. But a less data-driven task lets the CNN+optimization “show us its priors.”

We could also turn this into a generative model for images, perhaps by randomizing the choice of support surfaces. My point is not that it is so different from generative models commonly used in statistics. In fact I want to make the opposite point, that these problems share much with learning problems and should be studied together.
Subgrid dynamics

Some equations, like Navier-Stokes at high Reynolds number, have complicated structure on all length scales. This makes numerical simulations very expensive. On the other hand, we often do not care about the details of the solution at small scales, we would be happy if the simulation could somehow use a summary of its effects on the large scale solution.

There is a general philosophy of “effective field theory” which proceeds by solving for or “integrating out” the small distance dynamics. This works well in particle physics problems where at small distances one can create new particles, and conversely one can summarize their effects on large scales by changing the parameters and/or adding new terms to the equations of motion (renormalization). This is hard for Navier-Stokes. But maybe it can be done by using ML to learn the renormalization from small scale simulations.
Subgrid dynamics

Some equations, like Navier-Stokes at high Reynolds number, have complicated structure on all length scales. This makes numerical simulations very expensive. On the other hand, we often do not care about the details of the solution at small scales, we would be happy if the simulation could somehow use a summary of its effects on the large scale solution.

There is a general philosophy of “effective field theory” which proceeds by solving for or “integrating out” the small distance dynamics. This works well in particle physics problems where at small distances one can create new particles, and conversely one can summarize their effects on large scales by changing the parameters and/or adding new terms to the equations of motion (renormalization). This is hard for Navier-Stokes. But maybe it can be done by using ML to learn the renormalization from small scale simulations.
An example of this program can be seen in Bar-Sinai et al, PNAS 2019. They look at the 1D Burger’s equation

\[
\frac{\partial v}{\partial t} + \frac{\partial}{\partial x} \left( \frac{v^2}{2} - \eta \frac{\partial v}{\partial x} \right)^2 = f(x, t)
\]

Solutions of Burger’s equation develop shocks with large \(\frac{\partial v}{\partial t}\), which can only be represented by very fine grids and/or adaptive methods.

In this work one instead replaces the “naive” discretization

\[
\partial_x v(x) \sim (v(x + \epsilon) - v(x))/\epsilon
\]

by a learned “stencil”

\[
\sum_n \alpha_n v(x + n\epsilon).\]
ML and the string landscape

A very different application of ML is to understanding the “landscape” of compactifications of string theory. String/M theory is formulated in ten or eleven space-time dimensions, and the dimensions we don’t see must form a small compact manifold. A lot of work has gone into classifying the possibilities and developing techniques for computing the resulting physical predictions. These are encoded by effective field theories, which contain the Standard Model and usually additional “Beyond the SM” physics. There are many well-motivated possibilities: low energy supersymmetry, extra generations of quarks and leptons, exotic matter of various types – but so far no clear guidance for which ones string theory prefers.

The string constructions involve many combinatorial choices. Estimates of the number of vacua range from $10^{500}$ to $10^{272,000}$ – although most of these choices do not have directly observable consequences.
A very different application of ML is to understanding the “landscape” of compactifications of string theory. String/M theory is formulated in ten or eleven space-time dimensions, and the dimensions we don’t see must form a small compact manifold. A lot of work has gone into classifying the possibilities and developing techniques for computing the resulting physical predictions. These are encoded by effective field theories, which contain the Standard Model and usually additional “Beyond the SM” physics. There are many well-motivated possibilities: low energy supersymmetry, extra generations of quarks and leptons, exotic matter of various types – but so far no clear guidance for which ones string theory prefers.

The string constructions involve many combinatorial choices. Estimates of the number of vacua range from $10^{500}$ to $10^{272,000}$ – although most of these choices do not have directly observable consequences.
It has been found that, even restricting attention to the constructions which are most likely to contain the SM, only a fraction of order $10^{-9}$ (one in a billion) agree with the current observational constraints (and are realistic in this sense).

We would like to sample from this subset and estimate the fraction of realistic vacua which predict each of the hypothesized BSM physics scenarios. Now there are other factors which enter the estimate of the probability of a given scenario, which depend on its effects in cosmology and on its effects on our likelihood to see that scenario (the anthropic principle). But in many cases there is no reason to expect either effect.

Many techniques for combinatorial search and optimization have been tried to search for realistic models. Recently several groups are trying ML and particularly reinforcement learning.
It has been found that, even restricting attention to the constructions which are most likely to contain the SM, only a fraction of order $10^{-9}$ (one in a billion) agree with the current observational constraints (and are realistic in this sense).

We would like to sample from this subset and estimate the fraction of realistic vacua which predict each of the hypothesized BSM physics scenarios. Now there are other factors which enter the estimate of the probability of a given scenario, which depend on its effects in cosmology and on its effects on our likelihood to see that scenario (the anthropic principle). But in many cases there is no reason to expect either effect.

Many techniques for combinatorial search and optimization have been tried to search for realistic models. Recently several groups are trying ML and particularly reinforcement learning.
It has been found that, even restricting attention to the constructions which are most likely to contain the SM, only a fraction of order $10^{-9}$ (one in a billion) agree with the current observational constraints (and are realistic in this sense).

We would like to sample from this subset and estimate the fraction of realistic vacua which predict each of the hypothesized BSM physics scenarios. Now there are other factors which enter the estimate of the probability of a given scenario, which depend on its effects in cosmology and on its effects on our likelihood to see that scenario (the anthropic principle). But in many cases there is no reason to expect either effect.

Many techniques for combinatorial search and optimization have been tried to search for realistic models. Recently several groups are trying ML and particularly reinforcement learning.
Halverson et al 1903.11616 used RL (the A3C method developed by Mnih et al 1602.01783) to find realistic type IIA orientifold compactifications. The main choices to be made are the embeddings of D-branes, which determine the gauge group and matter content of the effective theory. In brief, an embedding is a multiset of vectors $V \in \mathbb{Z}^6$, which must satisfy a set of constraints, some cubic equalities and some linear inequalities involving six latent variables.

The most important observables are the ranks of the gauge group and the numbers of matter fields of each charge. The gauge groups are each associated to a vector in the multiset, and the ranks are simply their multiplicities. The charges are associated with pairs of vectors, and the number of matter fields of charge $(V, V')$ is a cross-product $V \times V'$.

In the Standard Model, the gauge group is $\text{SU}(3) \times \text{SU}(2) \times \text{U}(1)$, so the ranks must include $(3, 2, 1)$. The quarks and leptons then have known charges. One can predict the number of generations as well as additional exotic matter.
Halverson et al 1903.11616 used RL (the A3C method developed by Mnih et al 1602.01783 to find realistic type IIA orientifold compactifications. The main choices to be made are the embeddings of D-branes, which determine the gauge group and matter content of the effective theory. In brief, an embedding is a multiset of vectors $V_1 \in \mathbb{Z}^6$, which must satisfy a set of constraints, some cubic equalities and some linear inequalities involving six latent variables.

The most important observables are the ranks of the gauge group and the numbers of matter fields of each charge. The gauge groups are each associated to a vector in the multiset, and the ranks are simply their multiplicities. The charges are associated with pairs of vectors, and the number of matter fields of charge $(V, V')$ is a cross-product $V \times V'$.

In the Standard Model, the gauge group is $SU(3) \times SU(2) \times U(1)$, so the ranks must include $(3, 2, 1)$. The quarks and leptons then have known charges. One can predict the number of generations as well as additional exotic matter.
Halverson et al 1903.11616 used RL (the A3C method developed by Mnih et al 1602.01783) to find realistic type IIa orientifold compactifications. The main choices to be made are the embeddings of D-branes, which determine the gauge group and matter content of the effective theory. In brief, an embedding is a multiset of vectors \( V_1 \in \mathbb{Z}^6 \), which must satisfy a set of constraints, some cubic equalities and some linear inequalities involving six latent variables.

The most important observables are the ranks of the gauge group and the numbers of matter fields of each charge. The gauge groups are each associated to a vector in the multiset, and the ranks are simply their multiplicities. The charges are associated with pairs of vectors, and the number of matter fields of charge \((V, V')\) is a cross-product \(V \times V'\).

In the Standard Model, the gauge group is \(SU(3) \times SU(2) \times U(1)\), so the ranks must include \((3, 2, 1)\). The quarks and leptons then have known charges. One can predict the number of generations as well as additional exotic matter.
Computational tools for mathematics

- Computer mathematics / symbolic algebra platforms
  - Mathematica, Wolfram Alpha
  - SageMath, CoCalc
  - Specialized systems: GAP, Pari, Singular, Macaulay2, ...
  - And more: see Wikipedia’s List of computer algebra systems.

- Software development tools
  - Interactive programming languages: R / S, MatLab, Python, Julia, ...
  - Repositories with version control: Git, Github
  - Software libraries: PyPI, MathWorks FileExchange, ...
  - Interactive development environments

- Mathematical databases
  - Online Encyclopedia of Integer Sequences
  - Atlas of Finite Groups, GAP libraries
  - Kreuzer-Skarke database of Calabi-Yau manifolds
  - See https://mathdb.mathhub.info/ for many more

We will discuss some of these entries in more detail below.
Interactive theorem verification

- Proof assistants:
  - Isabelle/HOL, HOL Light
  - Coq
  - Lean
  - Mizar, Metamath, ...

All of these have libraries of proven theorems = more mathematical databases.

- Automated provers, mostly for first order logic: E, Vampire, ...
- SAT and SMT solvers: Z3, Alt-Ergo, ...
- Flagship verified proofs:
  - The four-color theorem
  - The Feit-Thompson theorem
  - Hales’s proof of the Kepler conjecture
Mathematical search

It’s not clear that any of these are mature technologies, in the sense that we can only expect incremental improvements. But some of them are widely used. For example, almost everyone here uses search engines and computer typesetting (mostly \LaTeX), most of us read and write papers on arXiv and look at Wikipedia articles, some of us write blogs or Wikipedia articles, and many of us use Mathematica or SageMath, and write programs in Python.

Let’s give some examples to illustrate some experimental tools, the problems they solve, and their limitations. We will start with search. Many mathematicians, for example Tim Gowers, have emphasized the potential value of a mathematical search engine. What is this?

The idea is, we describe a mathematical concept, and get back a list of documents containing claims, explanations, proofs, algorithms, computer code, etc., all relevant to our query.
It’s not clear that any of these are mature technologies, in the sense that we can only expect incremental improvements. But some of them are widely used. For example, almost everyone here uses search engines and computer typesetting (mostly \LaTeX), most of us read and write papers on arXiv and look at Wikipedia articles, some of us write blogs or Wikipedia articles, and many of us use Mathematica or SageMath, and write programs in Python.

Let’s give some examples to illustrate some experimental tools, the problems they solve, and their limitations. We will start with search. Many mathematicians, for example Tim Gowers, have emphasized the potential value of a mathematical search engine. What is this?

The idea is, we describe a mathematical concept, and get back a list of documents containing claims, explanations, proofs, algorithms, computer code, etc., all relevant to our query.
The easy case is when we are looking up a theorem or definition with its own name. Thus, if we want to know more about the four square theorem, or Sobolev spaces, we can just type the name into a search engine, and get many relevant responses:
While this is very helpful, it is hardly the answer to all of our questions.

Suppose we don’t know the name?

Suppose we invent a new concept or prove a new theorem? Are we really sure it is new? Maybe somebody already discovered or proved it. So what do we search for?

More than most human pursuits, in mathematics we can state our claims in simple and “universal” ways. Suppose we develop a search engine to look for formulas (including logical expressions) in documents. This would be a start, but of course a concept could be described by many different formulas, require several formulas for its definition, etc.. One might wonder how well a textual search could work.

Let’s try it out. The web site https://www.searchonmath.com offers formula search. Could we find “prime numbers” just from the definition?
While this is very helpful, it is hardly the answer to all of our questions.

Suppose we don’t know the name?

Suppose we invent a new concept or prove a new theorem? Are we really sure it is new? Maybe somebody already discovered or proved it. So what do we search for?

More than most human pursuits, in mathematics we can state our claims in simple and “universal” ways. Suppose we develop a search engine to look for formulas (including logical expressions) in documents. This would be a start, but of course a concept could be described by many different formulas, require several formulas for its definition, etc. One might wonder how well a textual search could work.

Let’s try it out. The web site https://www.searchonmath.com offers formula search. Could we find “prime numbers” just from the definition?
While this is very helpful, it is hardly the answer to all of our questions.

Suppose we don’t know the name?

Suppose we invent a new concept or prove a new theorem? Are we really sure it is new? Maybe somebody already discovered or proved it. So what do we search for?

More than most human pursuits, in mathematics we can state our claims in simple and “universal” ways. Suppose we develop a search engine to look for formulas (including logical expressions) in documents. This would be a start, but of course a concept could be described by many different formulas, require several formulas for its definition, etc. One might wonder how well a textual search could work.

Let’s try it out. The web site https://www.searchonmath.com offers formula search. Could we find “prime numbers” just from the definition?
While this is very helpful, it is hardly the answer to all of our questions. Suppose we don’t know the name?

Suppose we invent a new concept or prove a new theorem? Are we really sure it is new? Maybe somebody already discovered or proved it. So what do we search for?

More than most human pursuits, in mathematics we can state our claims in simple and “universal” ways. Suppose we develop a search engine to look for formulas (including logical expressions) in documents. This would be a start, but of course a concept could be described by many different formulas, require several formulas for its definition, etc. One might wonder how well a textual search could work.

Let’s try it out. The web site https://www.searchonmath.com offers formula search. Could we find “prime numbers” just from the definition?
While this is very helpful, it is hardly the answer to all of our questions.

Suppose we don’t know the name?

Suppose we invent a new concept or prove a new theorem? Are we really sure it is new? Maybe somebody already discovered or proved it. So what do we search for?

More than most human pursuits, in mathematics we can state our claims in simple and “universal” ways. Suppose we develop a search engine to look for formulas (including logical expressions) in documents. This would be a start, but of course a concept could be described by many different formulas, require several formulas for its definition, etc.. One might wonder how well a textual search could work.

Let’s try it out. The web site https://www.searchonmath.com offers formula search. Could we find “prime numbers” just from the definition?
I know, this is the set $\{1\}$, but maybe this is close enough.

Type it in into https://www.searchonmath.com. The result:

About 8 results in 0.42 seconds.

$P := \{p \in \mathbb{N} : \forall c \in \mathbb{N}, (c | p)\}$

By definition, how is a prime number represented? ...
https://math.stackexchange.com/questions/548513
Even numbers can be easily represented as $2n\$. Odd numbers as $2n+1\$.
An exactly divisible operation can be defined as ...

$P = \forall n \in \mathbb{N} : \exists a \in S : \text{gcd}$

A finite set $S$ satisfying $\forall n \in \mathbb{N}\ldots$
https://math.stackexchange.com/questions/2051716
I'd like some help with this question. Suppose we have some finite set $S$
satisfying $\forall n \in \mathbb{N}\exists \ldots$
Pretty good! I didn’t think this would work. Let’s try a harder one.

\[ \{ f \in L^2(\mathbb{R}) \land \frac{df}{dx} \in L^2(\mathbb{R}) \} \]
Not as good, but at least we got names we could try in another search. And actually our previous good result was a bit of a cheat as it used the fact that the symbol $p$ is often used to denote a prime. Consider

$$\{ q \in \mathbb{N} : \forall n \in \mathbb{N} : n | q \land n > 1 \land n < q \}$$

About 9 results in 0.18 seconds.

Prove that $c_0$ is complete by showing that $c_0$ is... [Link](https://math.stackexchange.com/questions/2462750)

Question: How do I show that $c_0$, which is shorthand for $\mathbb{C}_0(\mathbb{N})$, is closed in $\mathbb{L}_{\infty}$? I've tried to wo...

Formalizing sentences in propositional logic Ans... [Link](https://math.stackexchange.com/questions/2273219)

I'm studying for an introductory mathematical logic exam. Could you help me with formalizing the following conditions in...
A better search engine would have to “anonymize” the variables and try out many substitutions. See Kohlhase et al’s work on MathWebSearch for some of the issues. But there are many limitations of any formula-based search:

- Many concepts cannot be described by a simple formula in terms of standard concepts. One must build them up using two or three formulas, or else use a complicated non-canonical formula.
- Even when one has a simple definition, to refer to the standard concepts, one has to know and use the right formulations for them. Even when concepts are standard, their formulations are often not.

An example:

- \( \text{normal}(H, G) := H \in \text{subgroups}(G) \land \forall g \in G, gHg^{-1} \simeq H \)
- \( \forall H \in \text{subgroups}(G) : \text{normal}(H, G) \Rightarrow H \simeq \text{trivial} \lor H \simeq G \)

Do we have to remember “normal”, or \( H \triangleleft G \), or ?
In the 50’s and 60’s, the pioneers of AI developed automatic theorem provers, which generate logical deductions and search for proofs of given logical statements. Concurrently, the subfield of formal methods was developed, in which computer programs were given precise semantics allowing them to be rigorously verified.

This is of great practical value, especially for programs (an airplane autopilot, a CPU floating point unit) where mistakes can be extremely expensive. Thus it has been pursued intensively for decades, the formal methods community is fairly large and well-funded, and most of the systems we cited (Isabelle and Coq, though not Mizar) have software verification as the primary application.
Interactive theorem verification

In the 50’s and 60’s, the pioneers of AI developed automatic theorem provers, which generate logical deductions and search for proofs of given logical statements. Concurrently, the subfield of formal methods was developed, in which computer programs were given precise semantics allowing them to be rigorously verified.

This is of great practical value, especially for programs (an airplane autopilot, a CPU floating point unit) where mistakes can be extremely expensive. Thus it has been pursued intensively for decades, the formal methods community is fairly large and well-funded, and most of the systems we cited (Isabelle and Coq, though not Mizar) have software verification as the primary application.
As a prototypical example of software verification, let us briefly discuss sorting a list into alphabetical order. In an algorithms course one learns that a list of $N$ elements can be sorted in worst case time $N \log N$, but the algorithms (quicksort, heapsort, ...) are a bit tricky. On the other hand, logically defining the problem of sorting is not difficult. In Coq we can say

\begin{verbatim}
Definition is_a_sorting_algorithm (f: list nat -> list nat) :=
    \forall al, Permutation al (f al) \land sorted (f al).

Definition sorted' (al: list nat) :=
    \forall i j, i < j < length al \rightarrow nth i al 0 \leq nth j al 0.

Inductive Permutation : list A -> list A -> Prop :=
    | perm_nil: Permutation [] []
    | perm_skip x l l' : Permutation l l' \rightarrow Permutation (x::l) (x::l')
    | perm_swap x y l : Permutation (y::x::l) (x::y::l)
    | perm_trans l l' l'' :
        Permutation l l' \rightarrow Permutation l' l'' \rightarrow Permutation l l''.
\end{verbatim}

In https://softwarefoundations.cis.upenn.edu/vfa-current/Sort.html one can see formally verified proofs that runnable sorting programs satisfy this specification.
Let’s look a bit at a mathematical example, the fundamental theorem of algebra. As we all know, this states that the field $\mathbb{C}$ of complex numbers is algebraically closed, in other words every nonconstant polynomial $f(z)$ has a root.

This claim can be easily formalized: in the Lean theorem proving language, we can say

```lean
lemma exists_root {f : polynomial C} (hf : 0 < degree f) : ∃z : C, f.eval z = 0 :=
```

followed by the proof.
Here is an informal proof. We start by assuming that $f$ has no zero, to get a contradiction. (A constructive proof exists but is longer.)

1. We first show that $|f(z)|$ attains its minimum at some point $z_0$. A polynomial goes to infinity at infinity, so the infimum of $f$ is contained in a closed bounded region $R$. Since $|f|$ is continuous, the image of $R$ is closed and bounded, so it contains its infimum.

2. Expand around the location $z_0$ of the minimum by writing
   \[ f(z) = f(z_0) + (z - z_0)^n g(z) \]
   for some polynomial $g(z)$ such that $g(z_0) \neq 0$.

3. Now, consider a small circle $z = z_0 + \delta e^{i\theta}$. If we neglect the variation of $g(z)$ and look at $F(z) = f(z_0) + (z - z_0)^n g(z_0)$, it is easy to show that $z_0$ cannot be a minimum of $|F(z)|$, since $(z - z_0)^n$ takes every possible phase.

4. Intuitively, we then want to choose $\delta$ small enough such that we can neglect the variation of $g(z)$, so $z_0$ cannot be a minimum of $|f(z)|$, a contradiction. After a bit of algebra, it turns out that $|g(z) - g(z_0)| < |g(z_0)| \forall |z - z_0| \leq \delta$ suffices.
The Lean version of this proof takes about 100 lines: here is part 1.

```lean
lemma exists_forall_abs_polynomial_eval_le (p : polynomial ℂ) :
  ∃ x, ∀ y, (p.eval x).abs ≤ (p.eval y).abs :=
if hp₀ : 0 < degree p
then let ⟨r, hr₀, hr⟩ := polynomial.tendsto_infinity complex.abs hp₀ ((p.eval 0).abs) in
  let ⟨x, hx₁, hx₂⟩ := exists_forall_le_of_compact_of_continuous (λ y, (p.eval y).abs)
    (continuous_abs.comp p.continuous_eval)
    (closed_ball 0 r) (proper_space.compact_ball _) _
    (set.ne_empty_iff_exists_mem.2 {0, by simp [le_of_lt hr₀]}) in
  {x, λ y, if hy : y.abs ≤ r then hx₂ y $ by simpa [complex.dist_eq] using hy
    else le_trans (hx₂ _ (by simp [le_of_lt hr₀])) (le_of_lt (hr y (lt_of_not_ge hy))))
else {p.coeff 0, by rw [eq_C_of_degree_le_zero (le_of_not_gt hp₀)]; simp}
```

- The statement being proved is on line 13 - it is clear and intuitive.
- The language is “computerese” - but this is a question of taste and one can display the same content in more math-friendly notations.
- Unlike the informal proof, we had to give many propositions their own names and calling conventions, which also hurts readability.
- Commands like `rw` and `simpa` are “tactics,” explicit instructions to the proof verifier. These are procedural and tricky to get right.
Today’s ITV systems incorporate many advances in logic, such as dependent type theory (in Coq and Lean), and many advances in computer science. But despite all this work, theorem verification is more akin to programming than to any of the traditional skills of a mathematical scientist. And the many differences with informal proofs which we just cited, while each fairly simple, add up. At present ITV is hard to learn and use.

Still, many people believe that formalization and verification is a central part of the relation between computers and mathematics. This even includes theoretical physicists and others for whom rigorous proof is not a primary goal. It is hard to get a computer to understand anything, and here is a way for it to “understand truth.” So how to use this?

- Definitions are easier to write than proofs: focus on these?
- Perhaps ITVs need more reasoning methods than deductive logic?
- Will machine learning and AI help?
Today’s ITV systems incorporate many advances in logic, such as dependent type theory (in Coq and Lean), and many advances in computer science. But despite all this work, theorem verification is more akin to programming than to any of the traditional skills of a mathematical scientist. And the many differences with informal proofs which we just cited, while each fairly simple, add up. At present ITV is hard to learn and use.

Still, many people believe that formalization and verification is a central part of the relation between computers and mathematics. This even includes theoretical physicists and others for whom rigorous proof is not a primary goal. It is hard to get a computer to understand anything, and here is a way for it to “understand truth.” So how to use this?

- Definitions are easier to write than proofs: focus on these?
- Perhaps ITVs need more reasoning methods than deductive logic?
- Will machine learning and AI help?
Today’s ITV systems incorporate many advances in logic, such as dependent type theory (in Coq and Lean), and many advances in computer science. But despite all this work, theorem verification is more akin to programming than to any of the traditional skills of a mathematical scientist. And the many differences with informal proofs which we just cited, while each fairly simple, add up. At present ITV is hard to learn and use.

Still, many people believe that formalization and verification is a central part of the relation between computers and mathematics. This even includes theoretical physicists and others for whom rigorous proof is not a primary goal. It is hard to get a computer to understand anything, and here is a way for it to “understand truth.” So how to use this?

- Definitions are easier to write than proofs: focus on these?
- Perhaps ITVs need more reasoning methods than deductive logic?
- Will machine learning and AI help?
Tom Hales at U Pittsburgh has begun a project to create an online repository of formal abstracts, meaning statements of the main results of a mathematical paper, expressed in formal terms. Proofs would not be required, but it should be possible in principle to prove every abstract true or false. This project has many parts – here are a few (based on discussions with Tom):

- A solid ITV with dependent types – Lean.
- A library of standard concepts which can be used by abstracts, probably covering all of advanced undergraduate/early graduate level mathematics. A very rough estimate of the size is about 50,000 definitions filling 10,000 pages.
- Abstracts can be written in a controlled natural language which looks like standard mathematical text.
- Interactive tools to help search for, read and write abstracts.

While ambitious, such a system could be fully operational with its library in less than five years.
12 Sylow’s Theorems

In this section, let $G$ denote a fixed finite group.

In this section, let $gXg^{-1}$ stand for

$$\{gxg^{-1} \mid x \in X\},$$

where $(g : G)$ ($X : \text{set } G$).

**Definition 93** (conjugate). Assume that $(g : G)$. Assume that $H$ is a subgroup over $G$. The conjugate of $H$ by $g$ in $G$ is the subgroup $gHg^{-1}$ over $G$.

**Definition 94** (normalizer). Assume that $H$ is a subgroup over $G$. The normalizer of $H$ in $G$ is the subgroup $N$ over $G$ such that for all $x$, $x \in N \leftrightarrow xHx^{-1} = H$. This exists and is unique.

Let $|G|$ denote the order of $G$.

In this section, let $p$ denote a fixed prime number.

In this section, let $m$ denote the multiplicity of $p$ in $|G|$.

**Definition 95** (Sylow). A Sylow $p$ subgroup of $G$ is a subgroup $P$ over $G$ such that the order of $P$ is $p^m$. 
Definition 96. Let $\text{Syl}_p(G) = \{P \mid (P \text{ is a Sylow } p \text{ subgroup of } G)\}$.

Let $n_p(G)$ denote the size of $\text{Syl}_p(G)$. This is well subtyped (that is, there are finitely many Sylow $p$ subgroups).

Definition 97. Let $|N(p,G)|$ be equal to the size of the normalizer of each and every Sylow $p$ subgroup of $G$. This exists, is unique, and is well-defined.

Theorem 98 (Sylow1). There exists a Sylow $p$ subgroup of $G$.

Theorem 99 (Sylow 2). If $P, P'$ are Sylow $p$ subgroups of $G$ then there exists $(g : G)$ such that $P' = gPg^{-1}$.

Theorem 100 (Sylow 3a). Assume that $|G| = p'p^m$. We have $n_p(G)$ divides $p'$.

Theorem 101 (Sylow 3b). We have $p$ divides $(n_p(G) - 1)$.

Theorem 102 (Sylow 3c). We have $n_p(G) |N(p,G)| = |G|$.

Here is a sample of the controlled natural language Colada.
A formal proof is a chain of deductions, each based on one of a small fixed set of inference rules. Checking it is easy, the hard part is constructing one by choosing a useful deduction at each link. This is a problem in combinatorial search and has the usual exponential difficulty.

Deduction has received much attention from computer scientists, leading to the SAT/SMT solvers we discussed earlier. The next step up in generality is a first order prover, which can handle quantification over the ground universe, but not over predicates, functions, types, etc. The ITV s use higher order logic and complex type systems, which comes at an efficiency cost. Now, given a concrete goal, one can translate it into first order logic and apply an efficient prover. This “sledgehammer” approach has been quite effective.
Deductive logic is only one form of mathematical reasoning, and we all use many other forms. Some are rigorous, such as looking for counterexamples (more generally, using models). Others, such as analogy, are heuristic. There are also cases, say geometric reasoning, whose status is less clear. One can program computers to do geometric reasoning, but it is not clear to what extent this is like human reasoning.

All of this seems underdeveloped, in part because it can be hard to translate such arguments back into deductions. But as ITV is more broadly applied, there will be more interest in techniques which are convincing even if they don’t lead to deductive proofs.
As for the third question, clearly AI has made transformative progress over the last decade. Let’s come back to this after discussing some more applications.
Our discussion of mathematical search showed that textual search is limited, and that it is desirable to bring in considerations of meaning and correctness. This is the general topic of **semantic search**.

Semantic search has been a longstanding challenge for computer scientists and solving it in generality (unrestricted natural language queries about unstructured documents) might require AGI (artificial general intelligence). So,

1. Is mathematics more tractable for semantic search than other domains?
2. What data would the search engine have to work with?
3. What type of queries could we hope to handle?

More than most forms of communication, the meaning of mathematics can be made explicit in the text. Can we at least formulate semantic search in these terms?
Mathematical search engines

Our discussion of mathematical search showed that textual search is limited, and that it is desirable to bring in considerations of meaning and correctness. This is the general topic of **semantic search**.

Semantic search has been a longstanding challenge for computer scientists and solving it in generality (unrestricted natural language queries about unstructured documents) might require AGI (artificial general intelligence). So,

1. Is mathematics more tractable for semantic search than other domains?
2. What data would the search engine have to work with?
3. What type of queries could we hope to handle?

More than most forms of communication, the meaning of mathematics can be made explicit in the text. Can we at least formulate semantic search in these terms?
Mathematical search engines

Our discussion of mathematical search showed that textual search is limited, and that it is desirable to bring in considerations of meaning and correctness. This is the general topic of **semantic search**.

Semantic search has been a longstanding challenge for computer scientists and solving it in generality (unrestricted natural language queries about unstructured documents) might require AGI (artificial general intelligence). So,

1. Is mathematics more tractable for semantic search than other domains?
2. What data would the search engine have to work with?
3. What type of queries could we hope to handle?

More than most forms of communication, the meaning of mathematics can be made explicit in the text. Can we at least formulate semantic search in these terms?
As one considers more complicated queries with semantics, it becomes less and less likely that there will be a direct correspondence between the query and the contents of any of the target documents. Rather, one needs to say that the document contains text which is “relevant for” or “similar to” the query. This is expressed mathematically in terms of a \textbf{distance} function.

For example, to search for “cat and dog”, one might start with a word embedding, a map from words into a high dimensional vector space $V \cong \mathbb{R}^N$ with $N \sim 500$. Such embeddings are a standard part of NL processing (for example, \texttt{word2vec}).

Then, the logical connective “and” is expressed in terms of a function $f : V \times V \to V$, to get an embedding of the query $V_Q$. Finally, documents are assigned vectors $V_d$, perhaps by some $g : V^k \to V$ on keywords. Given a corpus these functions can be learned by training. Finally, the search procedure returns the documents which minimize the distance between $V_Q/\|V_Q\|$ and $V_d/\|V_d\|$.
As one considers more complicated queries with semantics, it becomes less and less likely that there will be a direct correspondence between the query and the contents of any of the target documents. Rather, one needs to say that the document contains text which is “relevant for” or “similar to” the query. This is expressed mathematically in terms of a distance function.

For example, to search for “cat and dog”, one might start with a word embedding, a map from words into a high dimensional vector space $V \cong \mathbb{R}^N$ with $N \sim 500$. Such embeddings are a standard part of NL processing (for example, word2vec).

Then, the logical connective “and” is expressed in terms of a function $f : V \times V \to V$, to get an embedding of the query $V_Q$. Finally, documents are assigned vectors $V_d$, perhaps by some $g : V^K \to V$ on keywords. Given a corpus these functions can be learned by training. Finally, the search procedure returns the documents which minimize the distance between $V_Q/\|V_Q\|$ and $V_d/\|V_d\|$.
As one considers more complicated queries with semantics, it becomes less and less likely that there will be a direct correspondence between the query and the contents of any of the target documents. Rather, one needs to say that the document contains text which is “relevant for” or “similar to” the query. This is expressed mathematically in terms of a **distance** function.

For example, to search for “cat and dog”, one might start with a word embedding, a map from words into a high dimensional vector space $V \cong \mathbb{R}^N$ with $N \sim 500$. Such embeddings are a standard part of NL processing (for example, *word2vec*).

Then, the logical connective “and” is expressed in terms of a function $f : V \times V \to V$, to get an embedding of the query $V_Q$. Finally, documents are assigned vectors $V_d$, perhaps by some $g : V^k \to V$ on keywords. Given a corpus these functions can be learned by training. Finally, the search procedure returns the documents which minimize the distance between $V_Q/\|V_Q\|$ and $V_d/\|V_d\|$.
Here’s a sketch of how one might define a distance between sets of logical statements, modeled on the concept of Gromov-Hausdorff distance between manifolds.

1. We start with two sets of statements $S$ and $S'$, each with a context: assignment of values or types to unbound variables, lists of prerequisite definitions, etc. If they don’t have a common context, rename the unbound variables and take the union of the prerequisites until they do.

2. Given a collection of sets $C \equiv \{S_i\}$ of statements $\{S_1, S_2, \ldots\}$, define the 1-ball of $C$ to be the collection of sets obtained by choosing a $S_i$ and adding a single statement produced by logical deduction. The $n$-ball $C_n$ is then the $n$’th iterate of this.

3. The distance between $S$ and $S'$ with a common context is the minimum $n$ such that $S \subseteq S'_n$ and $S' \subseteq S_n$.

While this distance is based on logic, one could replace steps 2 and 3 with vector embeddings.
CAR has attracted attention from mathematicians for its potential to solve some difficult challenges. One of the recognized challenges is “big proof,” the ability to work with, verify and communicate mathematical proofs which are too large for any individual to fully comprehend. The classic examples are proofs which involve case-by-case analysis of a vast number of cases, such as the proofs of the four-color theorem and of the Kepler conjecture.

In the proof of the four-color theorem (Appel and Haken 1976, Robertson et al 1995), one shows that every planar graph must contain as a subgraph, one out of a list of “reducible” configurations from which one can remove an edge while maintaining four-colorability. Thus, by induction, all graphs are four-colorable. And it turns out that such a list can consist of subgraphs with non-negative discrete curvature (arity) and a bound on the perimeter length, so they can be enumerated.
CAR has attracted attention from mathematicians for its potential to solve some difficult challenges. One of the recognized challenges is “big proof,” the ability to work with, verify and communicate mathematical proofs which are too large for any individual to fully comprehend. The classic examples are proofs which involve case-by-case analysis of a vast number of cases, such as the proofs of the four-color theorem and of the Kepler conjecture.

In the proof of the four-color theorem (Appel and Haken 1976, Robertson et al 1995), one shows that every planar graph must contain as a subgraph, one out of a list of “reducible” configurations from which one can remove an edge while maintaining four-colorability. Thus, by induction, all graphs are four-colorable. And it turns out that such a list can consist of subgraphs with non-negative discrete curvature (arity) and a bound on the perimeter length, so they can be enumerated.
This is a fine approach to proving the four-color theorem, except that the smallest known list has 633 subgraphs, and the computations required to prove its sufficiency are lengthy, even for a computer. (This list is not unique and perhaps the set of possible lists could be better understood.)

In 1611, Kepler conjectured that the most efficient (densest) way to pack spheres in three dimensions is the hexagonal close packing (and an infinite set of variations on it). This was proven in 1998 by Thomas Hales, with computer assistance. The proof involves solving over 100,000 linear programming problems.

Both of these proofs have been formally verified, meaning not just that the computations have been checked, but that the arguments by which these results imply the mathematical theorem have been formalized and verified. For the four-color theorem this was done in 2005 by Gonthier and collaborators (see Gonthier 2008 in the Notices of the AMS). For Kepler this was finished in 2014 by Hales and collaborators (the Flyspeck project).
This is a fine approach to proving the four-color theorem, except that the smallest known list has 633 subgraphs, and the computations required to prove its sufficiency are lengthy, even for a computer. (This list is not unique and perhaps the set of possible lists could be better understood.)

In 1611, Kepler conjectured that the most efficient (densest) way to pack spheres in three dimensions is the hexagonal close packing (and an infinite set of variations on it). This was proven in 1998 by Thomas Hales, with computer assistance. The proof involves solving over 100,000 linear programming problems.

Both of these proofs have been formally verified, meaning not just that the computations have been checked, but that the arguments by which these results imply the mathematical theorem have been formalized and verified. For the four-color theorem this was done in 2005 by Gonthier and collaborators (see Gonthier 2008 in the Notices of the AMS). For Kepler this was finished in 2014 by Hales and collaborators (the Flyspeck project).
This is a fine approach to proving the four-color theorem, except that the smallest known list has 633 subgraphs, and the computations required to prove its sufficiency are lengthy, even for a computer. (This list is not unique and perhaps the set of possible lists could be better understood.)

In 1611, Kepler conjectured that the most efficient (densest) way to pack spheres in three dimensions is the hexagonal close packing (and an infinite set of variations on it). This was proven in 1998 by Thomas Hales, with computer assistance. The proof involves solving over 100,000 linear programming problems.

Both of these proofs have been formally verified, meaning not just that the computations have been checked, but that the arguments by which these results imply the mathematical theorem have been formalized and verified. For the four-color theorem this was done in 2005 by Gonthier and collaborators (see Gonthier 2008 in the Notices of the AMS). For Kepler this was finished in 2014 by Hales and collaborators (the Flyspeck project).
There are also important theorems for which the proof, while composed of subarguments which can each be understood and checked by a human, in their totality strain the limits of human collaboration.

Perhaps the outstanding example is the classification of finite simple groups. This is a very central result in mathematics, whose full statement (including the definitions of the groups) can be made in perhaps 40 pages. According to Solomon 2018, the proof is spread over hundreds of articles, some of which depend on unpublished work. There is an ongoing project to publish a complete proof by 2023, consisting of 12 volumes, each many hundreds of pages long.

In physics and the exact sciences, although proofs are valuable, the central focus is on calculations whose results can be compared with experiment. Such calculations can be vast by any standard: millions of Feynman diagrams, thousands of particles or atoms, etc. And despite the efforts of multiple collaborations and scrutiny of reviewers, sometimes mistakes take a while to catch.
There are also important theorems for which the proof, while composed of subarguments which can each be understood and checked by a human, in their totality strain the limits of human collaboration.

Perhaps the outstanding example is the classification of finite simple groups. This is a very central result in mathematics, whose full statement (including the definitions of the groups) can be made in perhaps 40 pages. According to Solomon 2018, the proof is spread over hundreds of articles, some of which depend on unpublished work. There is an ongoing project to publish a complete proof by 2023, consisting of 12 volumes, each many hundreds of pages long.

In physics and the exact sciences, although proofs are valuable, the central focus is on calculations whose results can be compared with experiment. Such calculations can be vast by any standard: millions of Feynman diagrams, thousands of particles or atoms, etc. And despite the efforts of multiple collaborations and scrutiny of reviewers, sometimes mistakes take a while to catch.
I think you should use deep learning, here in Step 2
Many groups are applying machine learning to make ITV more automatic, in AITP projects such as TacticToe (Gauthier, Kalisczyk and Urban 2017), GamePad (Huang, Dhariwal, Song and Sutskever 2018), HOList (Bansal, Loos, Rabe, Szegedy and Wilcox 2019), CoqGym (Yang and Deng, 2019), and Proverbot9001 (Sanchez-Stern et al 2019).

In developing a proof, many choices must be made, including

- Premise selection. Out of the many known true statements, which ones should be used to make the next deduction? There could be 100’s of candidates in the current context, and if we search the entire library of proved theorems, millions of candidates.

- Tactic selection. Tactics include introduction of antecedent clauses, rewriting and simplification, and other simple logical steps. Modern ITV’s typically provide 40–100 tactics.

We can make an analogy between these successive choices, and the moves in a game of solitaire.
Many groups are applying machine learning to make ITV more automatic, in AITP projects such as TacticToe (Gauthier, Kaliszyk and Urban 2017), GamePad (Huang, Dhariwal, Song and Sutskever 2018), HOLList (Bansal, Loos, Rabe, Szegedy and Wilcox 2019), CoqGym (Yang and Deng, 2019), and Proverbot9001 (Sanchez-Stern et al 2019).

In developing a proof, many choices must be made, including

- Premise selection. Out of the many known true statements, which ones should be used to make the next deduction? There could be 100’s of candidates in the current context, and if we search the entire library of proved theorems, millions of candidates.

- Tactic selection. Tactics include introduction of antecedent clauses, rewriting and simplification, and other simple logical steps. Modern ITV’s typically provide 40–100 tactics.

We can make an analogy between these successive choices, and the moves in a game of solitaire.
A very successful approach to AI game play is reinforcement learning (RL), as famously used by AlphaGo. Its central parts are a pair of neural networks, one to choose moves and the other to evaluate the score of game positions. The original AlphaGo used a corpus of human games for initial training, and then generated games by self-play. (Another important element is Monte Carlo tree search – it turns out for Go that playing out a game many times with random moves, gives a good estimate for its score.)

Similarly, the AITP systems use a corpus of proven theorems as training data. As the verifier works through a theorem, each step of premise and tactic selection is saved, along with a summary of the state just before the choice is made. One can then use these pairs \((\text{state}, \text{selection})\) to train networks to do premise and tactic selection.

The better developed systems (Coq, HOL) have large libraries with 30,000–70,000 (short) theorems. This is enough training data to achieve success rates in proving similar theorems (a held-out testing set) of around 50%.
A very successful approach to AI game play is reinforcement learning (RL), as famously used by AlphaGo. Its central parts are a pair of neural networks, one to choose moves and the other to evaluate the score of game positions. The original AlphaGo used a corpus of human games for initial training, and then generated games by self-play. (Another important element is Monte Carlo tree search – it turns out for Go that playing out a game many times with random moves, gives a good estimate for its score.)

Similarly, the AITP systems use a corpus of proven theorems as training data. As the verifier works through a theorem, each step of premise and tactic selection is saved, along with a summary of the state just before the choice is made. One can then use these pairs (state, selection) to train networks to do premise and tactic selection.

The better developed systems (Coq, HOL) have large libraries with 30,000–70,000 (short) theorems. This is enough training data to achieve success rates in proving similar theorems (a held-out testing set) of around 50%.
A very successful approach to AI game play is reinforcement learning (RL), as famously used by AlphaGo. Its central parts are a pair of neural networks, one to choose moves and the other to evaluate the score of game positions. The original AlphaGo used a corpus of human games for initial training, and then generated games by self-play. (Another important element is Monte Carlo tree search – it turns out for Go that playing out a game many times with random moves, gives a good estimate for its score.)

Similarly, the AITP systems use a corpus of proven theorems as training data. As the verifier works through a theorem, each step of premise and tactic selection is saved, along with a summary of the state just before the choice is made. One can then use these pairs (state, selection) to train networks to do premise and tactic selection.

The better developed systems (Coq, HOL) have large libraries with 30,000–70,000 (short) theorems. This is enough training data to achieve success rates in proving similar theorems (a held-out testing set) of around 50%.
Auto-formalization

The total corpus of mathematical texts is much larger of course – there are about 1.5 million papers just on arXiv. Building on the success of neural networks for machine translation, could we develop a system which translates “informal” mathematical text to formal mathematics?

These experiments are in very early days, see for example https://arxiv.org/abs/1611.09703 by Kaliszyk et al. One problem is that there is no sizable corpus of aligned informal and formal mathematics to use as training data. So far this is dealt with by “informalizing” a formal corpus.

To my mind, a deeper problem is that mathematical texts are almost never self-contained, and a formalization cannot make much sense without the formal definitions of the concepts it refers to. In the best cases a text will only refer to standard concepts, so having a library as in Formal Abstracts would be a great help. But in many (most?) cases research papers refer to other research papers.
Auto-formalization

The total corpus of mathematical texts is much larger of course – there are about 1.5 million papers just on arXiv. Building on the success of neural networks for machine translation, could we develop a system which translates “informal” mathematical text to formal mathematics?

These experiments are in very early days, see for example https://arxiv.org/abs/1611.09703 by Kaliszyk et al. One problem is that there is no sizable corpus of aligned informal and formal mathematics to use as training data. So far this is dealt with by “informalizing” a formal corpus.

To my mind, a deeper problem is that mathematical texts are almost never self-contained, and a formalization cannot make much sense without the formal definitions of the concepts it refers to. In the best cases a text will only refer to standard concepts, so having a library as in Formal Abstracts would be a great help. But in many (most?) cases research papers refer to other research papers.
Auto-formalization

The total corpus of mathematical texts is much larger of course – there are about 1.5 million papers just on arXiv. Building on the success of neural networks for machine translation, could we develop a system which translates “informal” mathematical text to formal mathematics?

These experiments are in very early days, see for example https://arxiv.org/abs/1611.09703 by Kaliszyk et al. One problem is that there is no sizable corpus of aligned informal and formal mathematics to use as training data. So far this is dealt with by “informalizing” a formal corpus.

To my mind, a deeper problem is that mathematical texts are almost never self-contained, and a formalization cannot make much sense without the formal definitions of the concepts it refers to. In the best cases a text will only refer to standard concepts, so having a library as in Formal Abstracts would be a great help. But in many (most?) cases research papers refer to other research papers.
Could we mine arXiv to produce a “semi-formal” corpus – *i.e.* a list of definitions, equations, theorems which

- Are expressed in a single consistent formal language.
- Need not be provable or true, need not type check or pass other non-syntactic criteria.
- Treat each identifier as bound or unbound, put bound identifiers in sensible contexts (function definitions or quantified), and give unbound identifiers unique global names.

Getting the scoping right will be challenging. For example, if a document refers to a function $f$ many times, when are these the same function? I can’t think of any labeled data for this distinction so the system will have guess at it using textual and other cues. In any case this will only be accomplished up to some accuracy and we might want to work with multiple candidate interpretations. This step can be done document by document and produces a “local” semi-formal corpus.
Could we mine arXiv to produce a “semi-formal” corpus – *i.e.* a list of definitions, equations, theorems which

- Are expressed in a single consistent formal language.
- Need not be provable or true, need not type check or pass other non-syntactic criteria.
- Treat each identifier as bound or unbound, put bound identifiers in sensible contexts (function definitions or quantified), and give unbound identifiers unique global names.

Getting the scoping right will be challenging. For example, if a document refers to a function \( f \) many times, when are these the same function? I can’t think of any labeled data for this distinction so the system will have guess at it using textual and other cues. In any case this will only be accomplished up to some accuracy and we might want to work with multiple candidate interpretations.

This step can be done document by document and produces a “local” semi-formal corpus.
Once we can do this document-by-document semi-formalization, the next step is to

- Make the unbound identifiers globally aligned – if two expressions, even from different documents, refer to “the same” definition, then there is a good chance that the same token will be used in both.

This need not be perfect – for example two equivalent definitions might not be recognized as such – but it is crucial. The simplest approach to start with would be for the system to look for groups of documents with consistent notation. Perhaps some hand-crafted criteria would be worth making. More generally, besides comparing the way a definition is used in different documents, the system could pay attention to names and textual cues, and could use the citation graph.

Given a semi-formalized corpus, we could then produce a language model for formal mathematics, using the same techniques as for natural language models. For example, we could mask elements of a formula and try to predict them. This would give us an a priori global embedding to use for other tasks.
Once we can do this document-by-document semi-formalization, the next step is to

- Make the unbound identifiers globally aligned – if two expressions, even from different documents, refer to “the same” definition, then there is a good chance that the same token will be used in both.

This need not be perfect – for example two equivalent definitions might not be recognized as such – but it is crucial. The simplest approach to start with would be for the system to look for groups of documents with consistent notation. Perhaps some hand-crafted criteria would be worth making. More generally, besides comparing the way a definition is used in different documents, the system could pay attention to names and textual cues, and could use the citation graph.

Given a semi-formalized corpus, we could then produce a language model for formal mathematics, using the same techniques as for natural language models. For example, we could mask elements of a formula and try to predict them. This would give us an *a priori* global embedding to use for other tasks.
The big success of AlphaGo came from its ability to generate its own training data by self-play. Amazingly, it turned out that human games were not needed – AlphaGo Zero achieved superhuman skill without them. Could we make a theorem prover do self-play?

At first, one might say that to “win a game” is equivalent to proving a theorem. However this is simplistic as every step of a deduction proves a new logical statement. Somehow the results have to be scored according to how “significant,” “interesting” or “useful” the statement is, or how close the new statement is to a significant result.

Only rewarding the theorems considered interesting or significant by humans may not be giving the computer enough feedback. So, it may be necessary to give the computer its own ability to judge what is interesting. From an ML point of view this is just another scoring function which could be learned.
The big success of AlphaGo came from its ability to generate its own training data by self-play. Amazingly, it turned out that human games were not needed – AlphaGo Zero achieved superhuman skill without them. Could we make a theorem prover do self-play?

At first, one might say that to “win a game” is equivalent to proving a theorem. However this is simplistic as every step of a deduction proves a new logical statement. Somehow the results have to be scored according to how “significant,” “interesting” or “useful” the statement is, or how close the new statement is to a significant result.

Only rewarding the theorems considered interesting or significant by humans may not be giving the computer enough feedback. So, it may be necessary to give the computer its own ability to judge what is interesting. From an ML point of view this is just another scoring function which could be learned.
The big success of AlphaGo came from its ability to generate its own training data by self-play. Amazingly, it turned out that human games were not needed – AlphaGo Zero achieved superhuman skill without them. Could we make a theorem prover do self-play?

At first, one might say that to “win a game” is equivalent to proving a theorem. However this is simplistic as every step of a deduction proves a new logical statement. Somehow the results have to be scored according to how “significant,” “interesting” or “useful” the statement is, or how close the new statement is to a significant result.

Only rewarding the theorems considered interesting or significant by humans may not be giving the computer enough feedback. So, it may be necessary to give the computer its own ability to judge what is interesting. From an ML point of view this is just another scoring function which could be learned.
What is “interesting” mathematics? In Lenat’s AM system (1976) this was defined by hand-coded heuristics.

Perhaps given a large semi-formal corpus, this could be inferred by similarity of a statement to the statements in the corpus. After all, people try to make interesting statements and avoid uninteresting ones.

Rather than make an *a priori* definition of interesting, one can say that an interesting concept is one which aids reasoning. To the extent that the system can judge the complexity of its proofs, then a new statement which makes many proofs simpler is *ipso facto* interesting.

One could consider efficacy at more general tasks. Perhaps textbook problems would be a good source. As another example, given a mathematical definitions such as “finite group,” can the system take a pair of randomly chosen examples and efficiently prove that they are isomorphic or not isomorphic. As an even harder test, can the system enumerate groups with up to $k$ elements?
What is “interesting” mathematics? In Lenat’s AM system (1976) this was defined by hand-coded heuristics.

Perhaps given a large semi-formal corpus, this could be inferred by similarity of a statement to the statements in the corpus. After all, people try to make interesting statements and avoid uninteresting ones.

Rather than make an \textit{a priori} definition of interesting, one can say that an interesting concept is one which aids reasoning. To the extent that the system can judge the complexity of its proofs, then a new statement which makes many proofs simpler is \textit{ipso facto} interesting.

One could consider efficacy at more general tasks. Perhaps textbook problems would be a good source. As another example, given a mathematical definitions such as “finite group,” can the system take a pair of randomly chosen examples and efficiently prove that they are isomorphic or not isomorphic. As an even harder test, can the system enumerate groups with up to $k$ elements?
What is “interesting” mathematics? In Lenat’s AM system (1976) this was defined by hand-coded heuristics.

Perhaps given a large semi-formal corpus, this could be inferred by similarity of a statement to the statements in the corpus. After all, people try to make interesting statements and avoid uninteresting ones.

Rather than make an *a priori* definition of interesting, one can say that an interesting concept is one which aids reasoning. To the extent that the system can judge the complexity of its proofs, then a new statement which makes many proofs simpler is *ipso facto* interesting.

One could consider efficacy at more general tasks. Perhaps textbook problems would be a good source. As another example, given a mathematical definitions such as “finite group,” can the system take a pair of randomly chosen examples and efficiently prove that they are isomorphic or not isomorphic. As an even harder test, can the system enumerate groups with up to $k$ elements?
Summary

- Brief review of deep learning
- Machine learning in physics:
  - ML in physics simulations
  - ML and the string landscape
- Mathematical search
- AI for mathematical theorem proving

See also

- ML for neuroscience: work of Memming Park and others in SB Neuroscience.
- ML for protein folding: Laufer Center, CASP 13 results.