THOUGHTS ABOUT THE FUTURE
ML FOR PARTICLE PHYSICS

@KyleCranmer
New York University
Department of Physics
Center for Data Science
CILVR Lab
“Prediction is difficult, especially when dealing with the future”.

–Neils Bohr, Danish proverb, Yogi Berra, …
FRAMING

Where will our ML-powered future of physics be on the spectrum between:

• Basically the same strategy we use now, but faster, better, cheaper

• Completely reformulated, totally different, almost unrecognizable
  
  • e.g. end-to-end learning from sensors to discovery or new particles and new physical laws
We can leverage both the power of deep learning and inject our expert physics knowledge.

Discriminative or Generative?

- Advantages discriminative models:
  - Flexible map from input to target (low bias)
  - Efficient training algorithms available
  - Solve the problem you are evaluating on.
  - Very successful and accurate!

- Advantages generative models:
  - Inject expert knowledge
  - Model causal relations
  - Interpretable
  - Data efficient
  - More robust to domain shift
  - Facilitate un/semi-supervised learning
STATISTICAL FRAMING

**PREDICTION**

- Forward modeling
- Generation
- Simulation

- \( p(x, z | \theta, \nu) \)

**INFERENCE**

- Inverse problem
- Measurement
- Parameter estimation

\( \theta \)

- Parameters of interest

\( \nu \)

- Nuisance parameters

\( z \)

- Latent variables
- Monte Carlo truth

\( x \)

- Observed data
- Simulated data
Physics-Aware Machine Learning
Physics goes into the construction of a “Kernel” that defines the model

- Vocabulary of kernels + grammar for composition = powerful modeling

Structure Discovery in Nonparametric Regression through Compositional Kernel Search
David Duvenaud, James Robert Lloyd, Roger Grosse, Joshua B. Tenenbaum, Zoubin Ghahramani
International Conference on Machine Learning, 2013
pdf | code | poster | bibtex

Exploiting compositionality to explore a large space of model structures
Roger Grosse, Ruslan Salakhutdinov, William T. Freeman, Joshua B. Tenenbaum
Conference on Uncertainty in Artificial intelligence, 2012
pdf | code | bibtex

Mauna Loa atmospheric CO₂

Figure 4. First row: The posterior on the Mauna Loa dataset, after a search of depth 10. Subsequent rows show the yearly periodic structure. The third row, the scale has been changed in order to clearly show the yearly periodic structure.

\[ \text{Per} \times \text{SE} \times \text{RQ} \]
We can **inject** our knowledge of physics into the machine learning models! We can **extract** knowledge learned from the data!

**Physics-aware Gaussian Processes**

Final Kernel =

- Poisson fluctuations
- Mass Resolution
- Parton Density Functions
- Jet Energy Scale

[Correlation Matrix](http://bit.ly/2AkwYRG)

**QCD-Aware recursive neural networks**

ArXiv:1702.00748

**QCD-Aware graph convolutional neural networks**


\[
d_{ii'}^\alpha = \min(p_{ii}^{2\alpha}, p_{ii'}^{2\alpha}) \frac{\Delta R_{ii'}^2}{R^2}
\]
"There is great promise at the interface of physical models and data driven models."

–NEIL LAWRENCE

Representations / Embeddings
"LA MIA PARABOLA"

Figure by Federico Carminati, independent parallel inventions by Vincenzo Innocente & K.C.
Events $\sim 10^{15}$

partons $\sim 10$

hadrons $\sim 100$

detector design, alignment

sensors $10^8$

measured interactions with matter

calibration constants

theory parameters

momenta, particle type

energy deposit

sensor readout $10^8$

raw data

nuisance parameters

measured parton density, functions, etc.

Legend

- parameter of interest
- nuisance parameter
- latent variable
- observed covariate
- derived quantities
SIMULATION + RECONSTRUCTION

- Events ~$10^{15}$
  - Partons ~$10$
    - Momenta, Particle Type
  - Hadrons ~$10^{10}$
    - Momenta, Particle Type
  - Sensors $10^8$
    - Energy Deposit
  - Sensor Readout $10^8$
    - Raw Data

- Parameter Estimates, Likelihood, Posterior
  - Event-level Features
    - Jets ~$10$
      - Momenta, Summary Stats
    - Reconstructed Particles ~$10^6$
      - Momenta, Particle Type
    - Clusters ~$10^6$
      - Energy, Summary Stats
    - Tracks ~$10^6$
      - Momenta, Impact Parameter

Legend:
- Parameter of Interest
- Nuisance Parameter
- Latent Variable
- Observed Covariate
- Derived Quantities
- Measured Parton Density Functions, etc.
- Detector Design, Alignment
- Measured Interactions with Matter
- Calibration Constants
COMPOSITION & REDUCTIONISM

The traditional reconstruction algorithms can be seen as attempt to invert the generative process (point estimate / regression)

• generative model: $\theta \rightarrow z_1 \rightarrow z_2 \rightarrow z_2 \rightarrow x$

• Sequential Inversion: $x \rightarrow \hat{z}_3(x) \rightarrow \hat{z}_2(\hat{z}_3) \rightarrow \hat{z}_1(\hat{z}_2)$

Key points:

• can characterize & validate $p(\hat{z}_1 \mid z_1)$, $p(\hat{z}_2 \mid z_2)$, $p(\hat{z}_3 \mid z_3)$ with simulation

• these components are reusable (transfer learning)

  • e.g. an algorithm that looks for electrons in the data (segmentation & classification) and estimates their energy and momentum (regression).

• Provides a notion of “interpretable” that is practical and actionable

• Composition is at the heart of the reductionist paradigm of science
Economic growth has slowed down in recent years.

Das Wirtschaftswachstum hat sich in den letzten Jahren verlangsamt.

La croissance économique s'est ralentie ces dernières années.
JOINTLY OPTIMIZE HIERARCHICAL MODEL

particle embedding $\rightarrow$ jet embedding $\rightarrow$ event embedding $\rightarrow$ classifier

It scales!

arXiv:1702.00748
COMPOSITION OF REUSABLE COMPONENTS

How do these fit together?

Combine many of these ideas:

- **Large model**, but **sparsely activated**
- **Single model** to solve many tasks (100s to 1Ms)
- **Dynamically learn** and **grow pathways** through large model
- Hardware **specialized for ML supercomputing**
- **ML for efficient mapping** onto this hardware

---

[ML 2.0 at Google](https://drive.google.com/file/d/0B8z5oUp82DysZWF1RTFuX1NEZUk/view)
DIFFERENTIABLE REDUCTIONISM

The reconstruction algorithms can be seen as attempt to invert the generative process (point estimate / regression) sequentially

- generative model: $\theta \rightarrow z_1 \rightarrow z_2 \rightarrow z_2 \rightarrow x$

- Sequential Inversion: $x \rightarrow \hat{z}_3(x) \rightarrow \hat{z}_2(\hat{z}_3) \rightarrow \hat{z}_1(\hat{z}_2)$

Currently both generative model and inversion algorithms involve hand-engineered, code not developed for auto-diff / back propagation (effectively not differentiable)

- big gain from just reimplementing what we have in a **Differentiable Programming framework**

We can keep the compositional structure and gradually enhance each of the stages of the with deep learning components

- A high-level form of inductive bias (innate structure) on the networks

- jointly optimize & borrow power from all the tasks that use a certain component

  - maintain ability to characterize, validate, and interpret individual components

- transition from deterministic point estimate to probabilistic components for improved uncertainty estimation
Inductive Bias
IMPOSING STRUCTURE

Deep neural networks are a very flexible class of functions (they have high “capacity”)

- danger of overfitting the data, need lots of data for training, need regularization techniques (which implicitly impose some structure on the model)

If data has some (approximate) symmetry you want to take advantage of it — eg translational invariance.

- data augmentation (orbit of group operations)
- data preprocessing (mod out symmetry in data)
- impose symmetry in model (eg. conv nets, hard to generalize)

More generally, if there is some structure in the data (hierarchical structure, Markov process, ...) would like to take advantage of it.
Learning Physics with Deep Neural Networks

Can we learn physical properties from data? Machine learning offers a solution. It has many similarities with physics, requiring the approximation of functionals which depend on large numbers of variables, such as millions of pixels in images, letters in text, or particles in a physical system. Machine learning algorithms have considerably improved in the last 10 years through the processing of massive amounts of data. In particular, deep neural networks have spectacular applications, such as image classification and medical, industrial and physical data analysis.

In this lecture, Stéphane Mallat will show how machine learning can be applied to statistical physics, turbulent fluids and quantum chemistry. Beyond applications, he will highlight common mathematical approaches in physics and machine learning to overcome the issue of dimensionality. Two central pillars of such approaches are finding symmetries and separating phenomena at different scales. He will show that these pillars also govern the architecture and properties of deep convolutional neural networks.
Gauge Fields in Deep Learning

16 Apr 2019, 16:30
1h 30m
500-1-001 - Main Auditorium (CERN)

Description

Invited colloquium by Prof. Dr. Max Welling

Part of the IML workshop, but listed under the CERN colloquium category.

Joint work with Taco Cohen, Maurice Weiler and Berkay Kicanaoglu

ABSTRACT:
Gauge field theory is the foundation of modern physics, including general relativity and the standard model of physics. It describes how a theory of physics should transform under symmetry transformations. For instance, in electrodynamics, electric forces may transform into magnetic forces if we transform a static observer to one that moves at constant speed. Similarly, in general relativity acceleration and gravity are equated to each other under symmetry transformations. Gauge fields also play a crucial role in modern quantum field theory and the standard model of physics, where they describe the forces between particles that transform into each other under (abstract) symmetry transformations.

In this work we describe how the mathematics of gauge groups becomes inevitable when you are interested in deep learning on manifolds. Defining a convolution on a manifold involves transporting geometric objects such as feature vectors and kernels across the manifold, which due to curvature become path dependent. As such it becomes impossible to represent these objects in a global reference frame and one is forced to consider local frames. These reference frames are arbitrary and changing between them is called a (local) gauge transformation. Since we do not want our computations to depend on the specific choice of frames we are forced to consider equivariance of our convolutions under gauge transformations. These considerations result in the first fully general theory of deep learning on manifolds, with gauge equivariant convolutions as the necessary key ingredient.
“Of course, particle physicists are among the first to realize that nature is compositional.”

– YANN LECUN

“The world is compositional, or there is a god"

– STUART GEMAN
FIG. 3. Jet classification performance for various input representations of the RNN classifier, using $k_t$ topologies for the embedding. The plot shows that there is significant improvement from removing the image processing step and that significant gains can be made with more accurate measurements of the 4-momenta.

FIG. 4. Jet classification performance of the RNN classifier based on various network topologies for the embedding (particles scenario). This plot shows that topology is significant, as supported by the fact that results for $k_t$, C/A and desc-$p_T$ topologies improve over results for anti-$k_t$, as c-$p_T$ and random binary trees. Best results are achieved for C/A and desc-$p_T$ topologies, depending on the metric considered. Further supported by the poor performance of the random binary tree topology. We expected however that a simple sequence (represented as a degenerate binary tree) based on ascending and descending $p_T$ ordering would not perform particularly well, particularly since the topology does not use any angular information. Surprisingly, the simple descending $p_T$ ordering slightly outperforms the RNNs based on $k_t$ and C/A topologies. The descending $p_T$ network has the highest $p_T$ 4-momenta near the root of the tree, which we expect to be the most important. We suspect this is the reason that the descending $p_T$ outperforms the ascending $p_T$ ordering on particles, but this is not supported by the performance on towers. A similar observation was already made in the context of natural languages [24–26], where tree-based models have at best only slightly outperformed simpler sequence-based networks. While recursive networks appear as a principled choice, it is conjectured that recurrent networks may in fact be able to discover and implicitly use recursive compositional structure by themselves, without supervision.

d. Gating The last factor that we varied was whether or not to incorporate gating in the RNN. Adding gating increases the number of parameters to 48,761, but this is still about 20 times smaller than the number of parameters in the MaxOut architectures used in previous jet image studies. Table I shows the performance of the various RNN topologies with gating. While results improve significantly with gating, most notably in terms of $R_{\mu}=50\%$, the trends in terms of topologies remain unchanged.

e. Other variants Finally, we also considered a number of other variants. For example, we jointly trained a classifier with the concatenated embeddings obtained over $k_t$ and anti-$k_t$ topologies, but saw no significant performance gain. We also tested the performance of recursive activations transferred across topologies. For instance, we used the recursive activation learned with a $k_t$ topology when applied to an anti-$k_t$ topology and observed a significant loss in performance. We also considered particle and tower level inputs with an additional trimming preprocessing step, which was used for the jet image studies, but we saw a significant loss in performance. While the trimming degraded classification performance, we did not evaluate the robustness to pileup that motivates trimming and other jet grooming procedures.

B. Infrared and Collinear Safety Studies In proposing variables to characterize substructure, physicists have been equally concerned with classification performance and the ability to ensure various theoretical properties of those variables. In particular, initial work on jet algorithms focused on the Infrared-Collinear (IRC) safe conditions:

- **Infrared safety.** The model is robust to augmenting $e$ with additional particles $\{v_{N+1} \rightarrow \ldots \rightarrow v_N \}$ with $QCD$.
- **Generative process** is a tree-like, ~stationary Markov Process
- **Physics algorithms** exist to estimate the tree
- **Tree-RNN needs much less data to train!**

QCD-INSPIRED RECURSIVE NEURAL NETWORKS
**NETWORK IN NETWORK (1X1 CONVOLUTIONS)**

---

**Network In Network**

Min Lin\(^1\), Qiang Chen\(^2\), Shuicheng Yan\(^2\)

\(^1\)Graduate School for Integrative Sciences and Engineering
\(^2\)Department of Electronic & Computer Engineering
National University of Singapore, Singapore

\{linmin, chenqiang, eleyans\}@nus.edu.sg

---

**Embed each input:**

- **1x1 convolution**
- **network-in-network**

---

Map = 1x1 Convolution / Network in Network

"convolution" may be confusing in this context

- **Convolutional Neural Networks** are a close relative of map. A normal map applies a function to every element. Convolutional neural networks also look at neighboring elements, applying a function to a small window around every element.

Windowed Map = Convolutional Layer

Haskell: `zipWith a (tail xs)`

---

1x1 convolution image: https://www.youtube.com/watch?v=qVP574skyuM

arXiv:1312.4400
The Machine Learning Landscape of Top Taggers


---

### Table: Performance Metrics for Top Taggers

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>AUC</th>
<th>Acc</th>
<th>$1/\varepsilon_B$ ((\varepsilon_S = 0.3))</th>
<th>#Param</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Median</td>
<td>Single</td>
<td></td>
</tr>
<tr>
<td>CNN [16]</td>
<td>0.981</td>
<td>0.930</td>
<td>914±14</td>
<td>995±15</td>
</tr>
<tr>
<td>ResNeXt [30]</td>
<td>0.984</td>
<td>0.936</td>
<td>1122±47</td>
<td>1246±28</td>
</tr>
<tr>
<td>TopoDNN [18]</td>
<td>0.972</td>
<td>0.916</td>
<td>295±5</td>
<td>378±5</td>
</tr>
<tr>
<td>Multi-body N-subjettiness 6 [24]</td>
<td>0.979</td>
<td>0.922</td>
<td>792±18</td>
<td>802±12</td>
</tr>
<tr>
<td>Multi-body N-subjettiness 8 [24]</td>
<td>0.981</td>
<td>0.929</td>
<td>867±15</td>
<td>926±20</td>
</tr>
<tr>
<td>TreeNiN [43]</td>
<td>0.982</td>
<td>0.933</td>
<td>1025±11</td>
<td>1209±23</td>
</tr>
<tr>
<td>P-CNN</td>
<td>0.980</td>
<td>0.930</td>
<td>732±24</td>
<td>838±13</td>
</tr>
<tr>
<td>ParticleNet [47]</td>
<td>0.985</td>
<td>0.938</td>
<td>1298±46</td>
<td>1383±45</td>
</tr>
<tr>
<td>LBN [19]</td>
<td>0.981</td>
<td>0.931</td>
<td>836±17</td>
<td>852±67</td>
</tr>
<tr>
<td>LoLa [22]</td>
<td>0.980</td>
<td>0.929</td>
<td>722±17</td>
<td>768±11</td>
</tr>
<tr>
<td>Energy Flow Polynomials [21]</td>
<td>0.980</td>
<td>0.932</td>
<td>384</td>
<td></td>
</tr>
<tr>
<td>Energy Flow Network [23]</td>
<td>0.979</td>
<td>0.927</td>
<td>633±31</td>
<td>734±13</td>
</tr>
<tr>
<td>Particle Flow Network [23]</td>
<td>0.982</td>
<td>0.932</td>
<td>891±18</td>
<td>1005±21</td>
</tr>
<tr>
<td>GoaT</td>
<td>0.985</td>
<td>0.939</td>
<td>1368±140</td>
<td>1549±208</td>
</tr>
</tbody>
</table>
Comments on Deep Generative Models
LEARNING THE SIMULATOR

Noise $\sim N(0,1)$

Generative Model

http://torch.ch/blog/2015/11/13/gan.html

LEARNING THE SIMULATOR

Z

Noise $\sim N(0,1)$

Generative Model

$\rightarrow$

X

Figure 8: Average $\pi^+$ Geant shower (top), and average $\pi^+$ CaloGAN shower (bottom), with progressive calorimeter depth (left to right).

Figure 9: Five randomly selected $e^+$ showers per calorimeter layer from the training set (top) and the five nearest neighbors (by euclidean distance) from a set of CaloGAN candidates.

Figure 10: Five randomly selected $\gamma$ showers per calorimeter layer from the training set (top) and the five nearest neighbors (by euclidean distance) from a set of CaloGAN candidates.

Figure 11: Five randomly selected $\pi^+$ showers per calorimeter layer from the training set (top) and the five nearest neighbors (by euclidean distance) from a set of CaloGAN candidates.

http://torch.ch/blog/2015/11/13/gan.html

VARIATIONAL AUTO-ENCODER

Auto-Encoding Variational Bayes

[Kingma and Welling, 2013/2014]
[Rezende et al, 2014]

- \( q_\phi(z|x) = N(\mu, \sigma^2) \)
  
  \([\mu, \sigma^2] = f^{z|x}(x, \phi) = \text{multilayer neural net} \)

- Objective: lower bound of \( \log p(x) \).
  - Jointly optimized w.r.t. \( \phi \) and \( \theta \)
  - This is approx. maximum likelihood
  - Simple SGD:
    - Sampling small minibatches of data
    - Sampling from approx. posterior

- This also minimizes an expected KL divergence
  \( D_{KL}(q_\phi(z|x)||p(z|x)) \)
  
  \(-\) gives us cheap approx. inference for new datapoints

Kingma and Welling, Auto-encoding Variational Bayes, ICLR 2014
Rezende, Mohamed and Wierstra, Stochastic back-propagation and variational inference in deep latent Gaussian models, ICML 2014

Conv. net as encoder/decoder, trained on faces
Objective

- Consider the value function
  \[ V(D, G) = \mathbb{E}_{x \sim \mathbb{P}_{\text{data}}}[\log(D(x))] + \mathbb{E}_{z \sim \mathbb{P}_{\text{noise}}}[\log(1 - D(G(z)))]; \]

- We want to
  - For fixed \( G \), find \( D \) which maximizes \( V(D, G) \),
  - For fixed \( D \), find \( G \) which minimizes \( V(D, G) \);

- In other words, we are looking for the *saddle point*
  \[ (D^*, G^*) = \max_D \min_G V(D, G). \]
# GANs EVERYWHERE

## IML Machine Learning Working Group: special seminar on GANs

**Description**
Generative adversarial networks (GANs) are a recently introduced class of generative models, designed to produce realistic samples. They are relevant for High-Energy Physics to efficiently simulate events and detector interactions. First studies on calorimeter simulations promise of magnitude. Ian will present current research on improving GANs in g.

Ian Goodfellow is Research Scientist working in machine learning at Google, inventing generative adversarial networks and for writing the textbook.

There will be a coffee break before the seminar, starting at 14:00. For time for a general Q&A period, where students in particular are encouraged.

### Wed 17/4

<table>
<thead>
<tr>
<th>Time</th>
<th>Event</th>
<th>Speaker/Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>14:00</td>
<td>Coffee break</td>
<td></td>
</tr>
<tr>
<td>14:30</td>
<td>Seminar: Open challenges for improving Generative Adversarial Networks (GANs)</td>
<td>Speaker: Ian Goodfellow (Google)</td>
</tr>
<tr>
<td>15:15</td>
<td>Seminar questions</td>
<td></td>
</tr>
<tr>
<td>15:30</td>
<td>Break</td>
<td></td>
</tr>
<tr>
<td>16:00</td>
<td>General Q&amp;A</td>
<td></td>
</tr>
</tbody>
</table>

### Daily announcements

**500-1-001 - Main Auditorium, CERN**

**Applying Generative Models to Scientific Research**
Fedor Ratnikov
09:05 - 09:35

**DijetGAN: A Generative-Adversarial Network Approach for the Simulation of QCD Dijet Events at the LHC**
Serena Palazzo
09:35 - 09:55

**High Granularity Calorimeter Simulation using Generative Adversarial Networks**
Gul Rukh Khattak
09:55 - 10:15

**Deep generative models for fast shower simulation in ATLAS**
Aishik Ghosh
10:15 - 10:35

**Coffee break**
10:35 - 11:05

**Fast Simulation Using Generative Adversarial Network in LHCb**
Artem Maevskiy
11:05 - 11:25

**Model-Assisted GANs for the optimisation of simulation parameters and as an algorithm for fast Monte Carlo production**
Mr Saul Alonso Monsalve
11:25 - 11:45

**Event Generation and Statistical Sampling with Deep Generative Models**
Sydney Otten
11:45 - 12:05

**LUMIN - a deep learning and data science ecosystem for high-energy physics**
Giles Chatham Strong
12:05 - 12:25
Problem: the likelihood for GAN and VAE are intractable
Autoregressive models defined by

\[ p(x) = \prod_{t=1}^{T} p(x_t | x_{t-1}, \ldots, x_1). \]

have a tractable density. Train via maximum likelihood.
Autoregressive models defined by

\[ p(x) = \prod_{t=1}^{T} p(x_t \mid x_{t-1}, \ldots, x_1). \]

have a tractable density. Train via maximum likelihood.
Autoregressive models defined by

\[ p(x) = \prod_{t=1}^{T} p(x_t \mid x_{t-1}, \ldots, x_1). \]

have a tractable density. Train via maximum likelihood.
Approximations using Change-of-variables

Exploit the rule for change of variables for random variables:
- Begin with an initial distribution \( q_0(z_0|x) \).
- Apply a sequence of \( K \) invertible functions \( f_k \).

\[
q(z') = q(z) \left| \det \frac{\partial f}{\partial z} \right|^{-1}
\]

Sampling and Entropy
\[
z_K = f_K \circ \ldots \circ f_2 \circ f_1(z_0)
\]
\[
\log q_K(z_K) = \log q_0(z_0) - \sum_{k=1}^{K} \log \det \left| \frac{\partial f_k}{\partial z_k} \right|
\]

Distribution flows through a sequence of invertible transforms

[Rezende and Mohamed, 2015]
Glow: Generative Flow with Invertible 1×1 Convolutions

Diederik P. Kingma*, Prafulla Dhariwal*
OpenAI, San Francisco

(a) One step of our flow. (b) Multi-scale architecture (Dinh et al., 2016).

Table 3. Performance of Emerging convolutions on CIFAR10, ImageNet 32x32 and ImageNet 64x64 in bits per dimension (negative log₂-likelihood), and ± reports standard deviation.

<table>
<thead>
<tr>
<th></th>
<th>CIFAR10</th>
<th>ImageNet 32x32</th>
<th>ImageNet 64x64</th>
</tr>
</thead>
<tbody>
<tr>
<td>Real NVP</td>
<td>3.51</td>
<td>4.28</td>
<td>3.98</td>
</tr>
<tr>
<td>Glow</td>
<td>3.36 ±0.002</td>
<td>4.09</td>
<td>3.81</td>
</tr>
<tr>
<td>Emerging</td>
<td>3.34 ±0.002</td>
<td>4.09</td>
<td>3.81</td>
</tr>
</tbody>
</table>

https://arxiv.org/abs/1807.03039

Emerging Convolutions for Generative Normalizing Flows

Emiel Hoogeboom 1,2 Rianne van den Berg 1 Max Welling 1,3
FLOWS WITH CONTINUOUS TIME

**FFJORD: Free-form Continuous Dynamics for Scalable Reversible Generative Models**

Will Grathwohl*, Ricky T. Q. Chen*, Jesse Bettencourt†, Ilya Sutskever†, David Duvenaud†

---

**Figure 1:** FFJORD transforms a simple base distribution at \( t_0 \) into the target distribution at \( t_1 \) by integrating over learned continuous dynamics.

---

**Table 1:** A comparison of the abilities of generative modeling approaches.

<table>
<thead>
<tr>
<th>Change of Variables</th>
<th>Method</th>
<th>Train on data</th>
<th>One-pass Sampling</th>
<th>Exact log-likelihood</th>
<th>Free-form Jacobian</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Variational Autoencoders</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Generative Adversarial Nets</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td></td>
<td>Likelihood-based Autoregressive</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>Normalizing Flows</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>Reverse-NF, MAF, TAN</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td>NICE, Real NVP, Glow, Planar CNF</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td></td>
<td><strong>FFJORD</strong></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Can we use the idea for flows directly on an orthonormal basis of complex quantum wave functions?

- instead of $p(z) \rightarrow p(x)$ can we do $\phi_i(z) \rightarrow \psi_i(x)$?
- yes!

1) Start with:  
$$\int dz \, \phi_i(z) \phi_j^*(z) = \delta_{ij}$$

2) Change variables:  
$$\int dz \, \phi_i(z) \phi_j^*(z) = \int dx \left| \det \frac{\partial f}{\partial x} \right| \phi_i(f(x)) \phi_j^*(f(x)) = \delta_{ij}$$

3) Profit:  
$$\psi_i(x) = \phi_i(f(x)) \left| \det \frac{\partial f}{\partial x} \right|^\frac{1}{2}$$ complex! & orthonormal!
Unifying generative models and exact likelihood-free inference with conditional bijections

By Kyle Cranmer, Gilles Louppe

Recent work in density estimation uses a bijection $f : X \rightarrow Z$ (e.g. an invertible flow or autoregressive model) and a tractable density $p(z)$ (e.g. [1] [2] [3] [4]).

$$p(x) = p(f_\phi(x)) \left| \det \left( \frac{\partial f_\phi(x)}{\partial x_T} \right) \right|,$$

where $\phi$ are the internal network parameters for the bijection $f_\phi$. Learning proceeds via gradient ascent $\nabla_\phi \sum_i \log p(x_i)$ with data $x_i$ (i.e. maximum likelihood wrt. the internal parameters $\phi$). Since $f$ is invertible, then this model can also be used as a generative model for $X$.

This can be generalized to the conditional density $p(x|\theta)$ by utilizing a family of bijections $f_{\theta} : X \rightarrow Z$ parametrized by $\theta$ (e.g. [5] [6]).

$$p(x|\theta) = p(f_{\phi,\theta}(x)) \left| \det \left( \frac{\partial f_{\phi,\theta}(x)}{\partial x_T} \right) \right|$$

Here $\theta$ and $x$ are input to the network (and its inverse) and $\phi$ are internal network parameters. Again, learning proceeds via gradient ascent $\nabla_\phi \sum_i \log p(x_i|\theta_i)$ with data $x_i, \theta_i$.

We observe that not only can this model be used as a conditional generative model $p(x|\theta)$, but it can also be used to perform asymptotically exact, amortized likelihood-free inference on $\theta$.

This is particularly interesting when $\theta$ is identified with the parameters of an intractable, non-differentiable computer simulation or the conditions of some real world data collection process.

Comments

Many thanks to Durk Kingma, Max Welling, Ian Goodfellow, and Shakir Mohamed for enlightening discussions at NIPS2016.

Kyle Cranmer · 9 Dec, 2016
Parameter $\theta$ -> latent $z$ -> observable $x$ -> $r(x, z|\theta)$, $t(x, z|\theta)$ -> augmented data

arg min$_g L[g] \rightarrow \hat{r}(x|\theta) \rightarrow$ approximate likelihood ratio

Simulation -> Machine Learning -> Inference
Generative Models

“What I cannot create, I do not understand.”

—RICHARD FEYNMAN
Note, same NN can model birds, ants, volcanos, and calorimeters! Is that good or bad?
Correlation ≠ Causation
Mutual Information is a more general notion of ‘correlation’

\[ I(X; Y) = \sum_{y \in Y} \sum_{x \in X} p(x, y) \log \left( \frac{p(x, y)}{p_1(x) p_2(y)} \right), \]

- it is symmetric: \( I(X; Y) = I(Y; X) \)
- if and only if \( X, Y \) totally independent: \( I(X; Y) = 0 \)
- possible for \( X, Y \) to be uncorrelated, but not independent
SAME JOINT DISTRIBUTION, DIFFERENT CAUSAL MODEL

\[
x = \text{randn()}
\]
\[
y = x + 1 + \sqrt{3}\times\text{randn()}
\]

\[
y = 1 + 2\times\text{randn()}
\]
\[
x = (y-1)/4 + \sqrt{3}\times\text{randn()}/2
\]

\[
z = \text{randn()}
\]
\[
y = z + 1 + \sqrt{3}\times\text{randn()}
\]
\[
x = z
\]

https://colab.research.google.com/drive/1rjjA7teiZVHJCMTVD8KiZNu3EjS7Dmu#scrollTo=TlzzvcGOZdvW
CAUSATION > CORRELATION

\[ x = \text{randn()} \]
\[ x = 3 \]
\[ y = x + 1 + \sqrt{3} \cdot \text{randn()} \]
\[ x = 3 \]
\[ y = 1 + 2 \cdot \text{randn()} \]
\[ x = 3 \]
\[ x = (y-1)/4 + \sqrt{3} \cdot \text{randn()}/2 \]
\[ x = 3 \]
\[ z = \text{randn()} \]
\[ x = 3 \]
\[ x = z \]
\[ x = 3 \]
\[ y = z + 1 + \sqrt{3} \cdot \text{randn()} \]
\[ x = 3 \]

https://colab.research.google.com/drive/1rjjjA7teiZVHJCMTVD8KIZNu3EJ57Dmu#scrollTo=TlzzvcGOZdvW
1) We begin with Quantum Field Theory

2) Theory gives detailed prediction for high-energy collisions

hierarchical: $2 \rightarrow O(10) \rightarrow O(100)$ particles

3) The interaction of outgoing particles with the detector is simulated.

>100 million sensors
JUNIPR is a generative model for jets. Can train on real data!

**tractable likelihood**

\[
P_{\text{jet}}(\{p_1, \ldots, p_n\}) = \left[ \prod_{t=1}^{n-1} P_t(k_1^{(t+1)}, \ldots, k_{t+1}^{(t+1)} | k_1^{(t)}, \ldots, k_t^{(t)}) \right] \\
\times P_n(\text{end} | k_1^{(n)}, \ldots, k_n^{(n)}).
\]

... and it is interpretable

FAST SIM TO RECONSTRUCTED OBJECTS

Events $\sim 10^{15}$

- Partons $\sim 10$
  - Momenta, particle type

- Hadrons $\sim 100$
  - Momenta, particle type

- Sensors $10^8$
  - Energy deposit

- Sensor readout $10^8$
  - Raw data

- Nuisance parameters
  - Measured parton density functions, etc.
  - Detector design, alignment
  - Measured interactions with matter
  - Calibration constants

Parameter estimates, likelihood, posterior

Event-level features

- Jets $\sim 10$
  - Momenta, summary stats

Reconstructed particles $\sim 100$

- Clusters $\sim 100$
  - Energy, summary stats

- Tracks $\sim 100$
  - Momenta, impact parameter

Parameter of interest

Legend:
- Parameter of Interest
- Nuisance parameter
- Latent variable
- Observed covariate
- Derived quantities
FAST SIM TO RECONSTRUCTED OBJECTS

Events $\sim 10^{15}$

- Partons $\sim 10$
  - Momenta, particle type

- Hadrons $\sim 100$
  - Momenta, particle type

- Sensors $10^8$
  - Energy deposit

- Sensors readout $10^8$
  - Raw data

- Tracks $\sim 100$
  - Momenta, impact parameter

- Clusters $\sim 100$
  - Energy, summary stats

- Jets $\sim 10$
  - Momenta, summary stats

- Reconstructed particles $\sim 100$
  - Momenta, particle type

- Parameter estimates, likelihood, posterior

- Theory parameters

Legend:
- Parameter of Interest
- Nuisance parameter
- Latent variable
- Observed covariate
- Derived quantities

Correspondence:
- $\theta \leftrightarrow \hat{\theta}$
- $Z_1 \leftrightarrow \hat{Z}_3$
- $Z_2 \leftrightarrow \hat{Z}_2$
- $Z_3$
Learning with Non-Differentiable Simulators
Similar to GAN setup, but instead of using a neural network as the generator, use the actual simulation (eg. Pythia, GEANT).

Continue to use a neural network discriminator / critic.

**Difficulty**: the simulator isn’t differentiable, but there’s a trick!

Allows us to efficiently fit / tune simulation with stochastic gradient techniques!
Adversarial Variational Optimization of Non-Differentiable Simulators

Gilles Louppe, Joeri Hermans and Kyle Cranmer

Abstract

Complex computer simulators are increasingly used across fields of science as generative models tying parameters of an underlying theory to experimental observations. Inference in this setup is often difficult, as simulators rarely admit a tractable density or likelihood function. We introduce Adversarial Variational Optimization (AVO), a likelihood-free inference algorithm for fitting a non-differentiable generative model incorporating ideas from generative adversarial networks, variational optimization and empirical Bayes. We adapt the training procedure of generative adversarial networks by replacing the differentiable generative network with a domain-specific simulator. We solve the resulting non-differentiable minimax problem by minimizing variational upper bounds of the two adversarial objectives. Effectively, the procedure results in learning a proposal distribution over simulator parameters, such that the JS divergence between the marginal distribution of the synthetic data and the empirical distribution is minimized. We evaluate and compare the method with simulators producing both discrete and continuous data.

Likelihood-free inference

In scientific simulators, the likelihood of observations $\mathbf{x}$ given model parameters $\theta$ is usually implicitly defined as

$$p(\mathbf{x}|\theta) = \int p(\mathbf{x}|\mathbf{z}) p(\mathbf{z}|\theta) d\mathbf{z}.$$ 

This makes it intractable to evaluate.

In this work, our goal is to estimate the parameters $\theta$ that minimize the JS divergence between the (empirical) data distribution $p(\mathbf{x})$ and the implicit model $p(\mathbf{x}|\theta)$:

$$\theta^* = \arg \min_\theta \text{JS}(p(\mathbf{x}), p(\mathbf{x}|\theta)).$$

Examples. Particle physics, cosmology, climatology, epidemiology, computational topography.

$$L = -\frac{1}{N} \sum_{i=1}^{N} \log d_i + \lambda \mathcal{T}(\theta)$$

The case of particle physics. The Standard Model defines an implicit distribution $p(\mathbf{x}|\theta)$ from which high-dimensional observables can be simulated. Given data collected from Nature, we want to fit the model parameters $\theta$.

tl;dr.

1. Take the adversarial training setup of GANs.
2. Replace the generator network with a scientific simulator.
3. Bypass the non-differentiability with REINFORCE.

Illustration (particle physics)

Particle detector alignment. (Top and second row) Detector response for the detector offset $\mathbf{d} = 0 \times \mathbf{d}$ and $\mathbf{d} = 1$. These plots highlight the difficulty in observing a difference between samples from one or the other parameter setting. (Bottom) Training. In AVO, the discriminator adapts to the inference problem, regardless of its difficulty. It is not limited by the sub-optimality of an ad hoc summary statistic.

Tricks of the trade

Variational optimization/REINFORCE. Minimize variational upper bounds

$$U_q(\mathbf{x}) = \mathbb{E}_{\mathbf{z} \sim q(\mathbf{z}|\theta)} [L(\mathbf{z})]$$

defined by a proposal distribution $q(\mathbf{z}|\theta)$.

- Gradients $\nabla_\theta U_q$ are obtained with REINFORCE estimates, which only requires forward evaluations of the simulator $g$.
- This effectively results in minimizing

$$\text{JS}(p(\mathbf{x}), q(\mathbf{x}|\theta)).$$

where $q(\mathbf{x}|\theta)$ is obtained from $g(\theta, \mathbf{x})$.

Regularization (Mescheder et al, 2018). Penalty added to $U_q$ to improve convergence.

$$R_q(\theta) = \mathbb{E}_{\mathbf{z} \sim q(\mathbf{z}|\theta)} [||\nabla_{\theta} q(\mathbf{z}|\theta)||^2].$$

Benchmarks
Adversarial Variational Optimization of Non-Differentiable Simulators

Gilles Louppe, Joeri Hermans and Kyle Cranmer

Abstract

Complex computer simulators are increasingly used across fields of science as generative models tying parameters of an underlying theory to experimental observations. Inference in this setup is often difficult, as simulators rarely admit a tractable density or likelihood function. We introduce Adversarial Variational Optimization (AVO), a likelihood-free inference algorithm for fitting a non-differentiable generative model incorporating ideas from generative adversarial networks, variational optimization and empirical Bayes. We adapt the training procedure of generative adversarial networks by replacing the differentiable generative network with a domain-specific simulator. We solve the resulting non-differentiable minimax problem by minimizing variational upper bounds of the two adversarial objectives. Effective, the procedure results in learning a proposal distribution over simulator parameters, such that the JS divergence between the marginal distribution of the synthetic data and the empirical distribution is minimized. We evaluate and compare the method with simulators producing both discrete and continuous data.

Illustration (particle physics)

Tricks of the trade

Variational optimization/REINFORCE.

Minimize variational upper bounds

\[ U_d(\phi) = \mathbb{E}_{\theta \sim q(\theta|\psi)}[\mathcal{L}_d(\phi)] \]

\[ U_g(\psi) = \mathbb{E}_{\theta \sim q(\theta|\psi)}[\mathcal{L}_g(\theta)] \]

defined by a proposal distribution \( q(\theta|\psi) \).

- Gradients \( \nabla_\psi U_g \) are obtained with REINFORCE estimates, which only requires forward evaluations of the simulator \( g \).

- This effectively results in minimizing

\[ \text{JSD}(p_r(x), q(x|\psi)) \]

where \( q(x|\psi) = \int p(x|\theta)q(\theta|\psi)d\theta \).

\( R_1 \) regularization (Mescheder et al, 2018).

Penalty added to \( U_d \) to improve convergence.

\[ R_1(\phi) = \mathbb{E}_x p_r(x) \left[ ||\nabla_\phi d(x; \phi)||^2 \right] \]
MINING GOLD FROM THE SIMULATOR

- Computer simulation typically evolve along a tree-like structure of successive random branchings.

- The probabilities of each branching $p_i(z_i|z_{i-1}, \theta)$ are often clearly defined in the code:

  ```python
  if random() > 0.1 + 2.5 * model_parameter:
    do_one_thing()
  else:
    do_another_thing()
  ```

- For each run of the simulator, we can calculate the probability of the chosen path for different values of the parameters, and the "joint likelihood ratio":

  $$r(x, z|\theta_0, \theta_1) = \frac{p(x, z|\theta_0)}{p(x, z|\theta_1)} = \prod_i \frac{p(z_i|z_{i-1}, \theta_0)}{p(z_i|z_{i-1}, \theta_1)}$$

Similar to the joint likelihood ratio, from the simulator we can extract the joint score:

$$t(x, z|\theta_0) \equiv \nabla_{\theta} \log p(x, z_d, z_s, z_p|\theta) \bigg|_{\theta_0}$$
MINING GOLD FROM THE SIMULATOR

- Computer simulation typically evolve along a tree-like structure of successive random branchings.

- The probabilities of each branching $p_i(z_i|z_{i-1}, \theta)$ are often clearly defined in the code:

  ```python
  if random() > 0.1 + 2.5 * model_parameter:
      do_one_thing()
  else:
      do_another_thing()
  ```

- For each run of the simulator, we can calculate the probability of the chosen path for different values of the parameters, and the "joint likelihood ratio":

  $$r(x,z|\theta_0, \theta_1) = \frac{p(x,z|\theta_0)}{p(x,z|\theta_1)} = \prod_i \frac{p(z_i|z_{i-1}, \theta_0)}{p(z_i|z_{i-1}, \theta_1)}$$

Joint score are REINFORCE gradients
Where simulation viewed as policy
Connection to reinforcement learning!
LEARNING WITH AUGMENTED DATA

Parameter $\theta$

Latent $z$

Observable $x$

Augmented data

Simulation

Machine Learning

Inference

We have joint likelihood ratio

$$r(x,z|\theta_0, \theta_1) = \frac{p(x,z,x_d,z_s,z_p|\theta_0)}{p(x,z,x_d,z_s,z_p|\theta_1)}$$

With $r(x,z|\theta_0, \theta_1)$, we define the functional

$$L_r[p(x|\theta_0, \theta_1)] = \int dx \int dz p(x,z|\theta_1) \left( \frac{r(x|\theta_0, \theta_1) - r(x,z|\theta_0, \theta_1)}{r(x,z|\theta_0, \theta_1)} \right)^2$$

One can show it is minimized by

$$r(x|\theta_0, \theta_1) = \arg \min_{\tilde{r}(x|\theta_0, \theta_1)} L_r[\tilde{r}(x|\theta_0, \theta_1)]$$

We want likelihood ratio

$$r(x|\theta_0, \theta_1) = \frac{p(x|\theta_0)}{p(x|\theta_1)}$$

Similar to the joint likelihood ratio, we can calculate the joint score

$$t(x,z|\theta_0) \equiv \nabla_{\theta} \log p(x,z,x_d,z_s,z|\theta)$$

Given $t(x,z|\theta_0)$, we define the functional

$$L_t[\tilde{t}(x|\theta_0)] = \int dx \int dz p(x,z|\theta_0) \left( \frac{t(x|\theta_0) - t(x,z|\theta_0)}{t(x,z|\theta_0)} \right)^2$$

One can show it is minimized by

$$t(x|\theta_0) = \arg \min_{\tilde{t}(x|\theta_0)} L_t[\tilde{t}(x|\theta_0)]$$

We want score

$$t(x|\theta_0) \equiv \nabla_{\theta} \log p(x|\theta)$$

Again, we implement this minimization through machine learning

[J. Brehmer, K. Cranmer, G. Louppe, J. Pavez
1805.00013, 1805.00020, 1805.12244]
Spectrum of function approximation

<table>
<thead>
<tr>
<th>Classes</th>
<th>Compositional Structure</th>
<th>Function Primitives</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fully Known</td>
<td>Known</td>
<td>Known</td>
<td>No learning necessary</td>
</tr>
<tr>
<td>Partially Known</td>
<td>Known</td>
<td>Some known</td>
<td></td>
</tr>
<tr>
<td>Structure only</td>
<td>Known</td>
<td>Unknown</td>
<td></td>
</tr>
<tr>
<td>Primitives only</td>
<td>Unknown</td>
<td>Known</td>
<td></td>
</tr>
<tr>
<td>Unknown</td>
<td>Unknown</td>
<td>Unknown</td>
<td>1. Compositional structure may be expected, vs. 2. Nothing assumed</td>
</tr>
</tbody>
</table>

NN Approximation of a primitive function

- Wrap a (un)known function
- The wrapper contains both the known function and a NN that approximates it
- Intermediate loss is defined as the discrepancy between the known function and the NN
- Output may be either the known function's and NN's output

Slides from Kyunghyun Cho at DataScience @ HEP 2017
Simulation as Causal Generative Model
Facilitates inference

(defquery arrange-bumpers []
    (let [number-of-bumpers (sample (poisson 20))
          bumpydist (uniform-continuous 0 10)
          bumpxdist (uniform-continuous -5 14)
          bumper-positions (repeatedly
                              number-of-bumpers
                              #(vector (sample bumpxdist)
                                       (sample bumpydist))
          ]
     ;; code to simulate the world
     world (create-world bumper-positions)
     end-world (simulate-world world)
     balls (:balls end-world)
     
     ;; how many balls entered the box?
     num-balls-in-box (balls-in-box end-world)
     
     obs-dist (normal 4 0.1))
    )

(observedobs-dist num-balls-in-box)
Simulation as Causal Generative Model
Facilitates inference

(defquery arrange-bumpers []
  (let [number-of-bumpers (sample (poisson 20))
        bumpydist (uniform-continuous 0 10)
        bumpxdist (uniform-continuous -5 14)
        bumper-positions (repeatedly
                           number-of-bumpers
                           #(vector (sample bumpxdist)
                                    (sample bumpydist)))]
    ;; code to simulate the world
    world (create-world bumper-positions)
    end-world (simulate-world world)
    balls (:balls end-world)
    ;; how many balls entered the box?
    num-balls-in-box (balls-in-box end-world)
    obs-dist (normal 4 0.1)]
  (observe obs-dist num-balls-in-box)
Probabilistic programming languages (PPLs) attempt to decouple inference algorithms from model building, by creating a simple, yet expressive, syntax that allows one to take advantage of these powerful inference algorithms on any probabilistic generative model expressed as a regular computer program. Universal PPLs allow the expression of unrestricted probability models in a Turing-complete fashion [35–37], and there is a recent trend in combining these with variational inference and deep learning, leading to tools such as Pyro [38], ProbTorch [39], and Edward [40]. This is in contrast to languages such as Stan [41] that target the more restricted model class of probabilistic graphical models [26].

It is interesting to think of non-standard interpretation of the simulator code to compile a non-standard output

• eg. Output of compiler is a graphical model

• eg. Output of compiler is derivative of function

• eg. Output of compiler is joint score
**PROBABILISTIC PROGRAMMING**

**Idea:** hijack the random number generators and use Neural Network to perform a very fancy type of importance sampling

- Neural Network powered inference engine (python)
- real-world scientific simulator (C++)

**Simulator C++**

Pythia / Sherpa / GEANT / …

**Observation**

**Mean Simulated Observation**

NERSC, Lawrence Berkeley National Lab

arXiv:1807.07706
Interpretability

Latent probabilistic structure of 250 most frequent trace types

px  pz  Rejection sampling

py  Decay

(a) Prior execution $p(x)$.

Calorimeter

Rejection

(b) Posterior execution $p(x|y)$ conditioned on a given calorimeter observation $y$. 

[slide: Atılım Güneş Baydin]
arXiv:1807.07706
HAVE A PLAN TO TELL YOU WHERE YOU ARE GOING

GOALS

DEFINE THE MILESTONES THAT WILL ASSIST WITH REACHING YOUR GOALS

OBJECTIVES

DECIDE THE PLAN OF ACTION TO ACHIEVE YOUR OBJECTIVES

STRATEGIES

IDENTIFY THE TOOLS YOU WILL USE TO IMPLEMENT YOUR STRATEGIES

TACTICS
GOALS & STRATEGIES

Goals:
- Use machine learning to do better science

Strategies:
- Import domain knowledge into models (inductive bias)
- Export knowledge from learned models
- Leverage machine learning for intractable inverse problems
- Incorporate traditional scientific concerns into the learning paradigm
  - include impact of domain shift / systematics uncertainties into objective
  - maintain an actionable, scientifically-useful notion of “interpretability”
  - use real-world data for training when possible
  - be data efficient
- Modify codebase to facilitate use of these techniques
GOALS & STRATEGIES

**Goals:**

- Use machine learning to do better science

**Strategies:**

- Import domain knowledge into models (inductive bias)
- Export knowledge from learned models
- Leverage machine learning for intractable inverse problems
- Incorporate traditional scientific concerns into the learning paradigm
  - include impact of domain shift / systematics uncertainties into objective
  - maintain an actionable, scientifically-useful notion of “interpretability”
  - use real-world data for training when possible
  - be data efficient
- Modify codebase to facilitate use of these techniques

POSED AS QUESTIONS, THESE STRATEGIES DEFINE INTERESTING RESEARCH DIRECTIONS.
STRATEGIES & TACTICS

**Strategy:** Import domain knowledge into models

- **Tactic:** exploit symmetries in the data
- **Tactic:** exploit geometric structure of the data
- **Tactic:** exploit causal structure of the generative process
- **Tactic:** exploit hierarchical / compositional structure
- **Tactic:** exploit Markov property of the generative process
- **Tactic:** exploit tangent space of statistical manifold

**Strategy:** Export knowledge from learned models

- **Tactic:** learnable components that can be interpreted

**Strategy:** maintain an actionable, scientifically-useful notion of “interpretability”

- **Tactic:** compose model from reusable components that perform a specific task and can be individually characterized & validated
STRATEGIES & TACTICS

Strategy: Leverage machine learning for intractable inverse problems

- **Tactic**: Use the likelihood ratio trick to convert a discriminative classifier into a density ratio
- **Tactic**: Use autoregressive models & normalizing flows for conditional density estimation
- **Tactic**: Use universal probabilistic programming
- **Tactic**: Approximate gradients of non-differentiable, black-box models (AVO, RELAX, …)

Strategy: Include impact of systematics uncertainties into objective

- **Tactic**: Design loss functions more relevant to scientific goals
- **Tactic**: Adversarial training for continuous domain adaptation & fairness (“learning to pivot”)

Strategy: Use real-world data for training when possible

- **Tactic**: Weakly supervised learning

Strategy: Be data efficient

- **Tactic**: Exploiting domain knowledge can dramatically reduce number of parameters
CONCLUSION

My prediction is that the future of ML+physics **should** be a hybrid

- leverage expert knowledge of the generative process
- Inductive bias: compositionally, markov
- learn surrogates that extract relevant features for inference task

Several strategies to incorporate domain knowledge into the model

- starting point: migrate current code bases to differentiable and/or probabilistic programming framework
- gradually replace components with deep learning

Helpful to establish more actionable notions of “interpretability”
Deep Learning → Differentiable Functional Programming

“Representations are Types”

– Chris Olah
READ THIS!

http://colah.github.io/posts/2015-09-NN-Types-FP/

• **Encoding Recurrent Neural Networks** are just folds. They’re often used to allow a neural network to take a variable length list as input, for example taking a sentence as input.

```
fold = Encoding RNN
Haskell: foldl a s
```

• **Generating Recurrent Neural Networks** are just unfolds. They’re often used to allow a neural network to produce a list of outputs, such as words in a sentence.

```
unfold = Generating RNN
Haskell: unfoldr a s
```

• **General Recurrent Neural Networks** are accumulating maps. They’re often used when we’re trying to make predictions in a sequence. For example, in voice recognition, we might wish to predict a phoneme for every time step in an audio segment, based on past context.

```
Accumulating Map = RNN
Haskell: mapAccum r s
```

• **Bidirectional Recursive Neural Networks** are a more obscure variant, which I mention primarily for flavor. In functional programming terms, they are a left and a right accumulating map zipped together. They’re used to make predictions over a sequence with both past and future context.

```
Zipped Left & Right Accumulating Map = Bidirectional RNN
Haskell: zip (mapAccum r s x) (mapAccum r' s' x')
```

• **Convolutional Neural Networks** are a close relative of map. A normal map applies a function to every element. Convolutional neural networks also look at neighboring elements, applying a function to a small window around every element.

```
Windowed Map = Convolutional Layer
Haskell: zipWith a x (tail x)
```

Two dimensional convolutional neural networks are particularly notable. They have been behind recent successes in computer vision. (More on conv nets.)

```
Two Dimensional Convolutional Network
```

• **Recursive Neural Networks** (“TreeNets”) are catamorphisms, a generalization of folds. They consume a data structure from the bottom up. They’re mostly used for natural language processing, to allow neural networks to operate on parse trees.

```
Catamorphism = TreeNet
Haskell: cata a
```

Hype: Compositional Machine Learning and Hyperparameter Optimization

Hype is a proof-of-concept deep learning library, where you can perform optimization on compositional machine learning systems of many components, even when such components themselves internally perform optimization.

This is enabled by nested automatic differentiation (AD) giving you access to the automatic exact derivative of any floating-point value in your code with respect to any other. Underlying computations are run by a BLAS/LAPACK backend (OpenBLAS by default).

Automatic derivatives

You do not need to worry about supplying gradients (or Hessians) of your models, which are computed exactly and efficiently by AD. The underlying AD functionality is provided by DiffSharp.

"Reverse mode" AD is a generalized form of "backpropagation" and is distinct from numerical or symbolic differentiation.

In addition to reverse AD, Hype makes use of forward AD and nested combinations of forward and reverse AD. The core differentiation API provides gradients, Hessians, Jacobians, directional derivatives, and matrix-free exact Hessian- and Jacobian-vector products.

Hypergradients

You can get exact gradients of the training or validation loss with respect to hyperparameters. These hypergradients allow you to do gradient-based optimization of gradient-based optimization, meaning that you can do things like optimizing learning rate and momentum schedules, weight initialization parameters, or step sizes and mass matrices in Hamiltonian Monte Carlo models. (A recent article doing this with Python: Maclaurin, Dougal, David Duvenaud, and Ryan P. Adams. "Gradient-based Hyperparameter Optimization through Reversible Learning." arXiv preprint arXiv:1502.03492 (2015).)

```hype
1: open Hype
2: open Hype.Neural
3: 
4: // Train a network with stochastic gradient descent and a learning rate schedule
5: let train (x:DV) =
6:     let n = FeedForward()
7:     n.Add(Linear(784, 300))
8:     n.Add(tanh)
9:     n.Add(Linear(300, 10))
```
Sparsity
Compressed sensing

From Wikipedia, the free encyclopedia

Compressed sensing (also known as compressive sensing, compressive sampling, or sparse sampling) is a signal processing technique for efficiently acquiring and reconstructing a signal, by finding solutions to underdetermined linear systems. This is based on the principle that, through optimization, the sparsity of a signal can be exploited to recover it from far fewer samples than required by the Shannon-Nyquist sampling theorem. There are two conditions under which recovery is possible. The first one is sparsity which requires the signal to be sparse in some domain. The second one is incoherence which is applied through the isometric property which is sufficient for sparse signals.[2][3]

Overview

A common goal of the engineering field of signal processing is to reconstruct a signal from a series of sampling measurements. In general, this task is impossible because there is no way to reconstruct a signal during the times that the signal is not measured. Nevertheless, with prior knowledge or assumptions about the signal, it turns out to be possible to perfectly reconstruct a signal from a series of measurements (acquiring this series of measurements is called sampling). Over time, engineers have improved their understanding of which assumptions are practical and how they can be generalized.

An early breakthrough in signal processing was the Nyquist–Shannon sampling theorem. It states that if a real signal's highest frequency is less than half of the sampling rate (or less than the sampling rate, if the signal is complex), then the signal can be reconstructed perfectly by means of sinc interpolation. The main idea is that with prior knowledge about constraints on the signal's frequencies, fewer samples are needed to reconstruct the signal.

Around 2004, Emmanuel Candès, Justin Romberg, Terence Tao, and David Donoho proved that given knowledge about a signal's sparsity, the signal may be reconstructed with even fewer samples than the sampling theorem requires.[4][5] This idea is the basis of compressed sensing.
Deep Decoder: Concise Image Representations from Untrained Non-convolutional Networks

Reinhard Heckel∗ and Paul Hand†

∗Dept. of Electrical and Computer Engineering, Rice University
†Dept. of Mathematics and College of Computer and Information Science, Northeastern University

March 12, 2019

Data models are central for signal and image processing and play a key role in compression and inverse problems such as denoising, super-resolution, and compressive sensing. These data models impose structural assumptions on the signal or image, which are traditionally based on expert knowledge. For example, imposing the assumption that an image can be represented with few non-zero wavelet coefficients enables modern (lossy) image compression [Ant+92] and efficient denoising [Don95].

In recent years, it has been demonstrated that for a wide range of imaging problems, from compression to denoising, deep neural networks trained on large datasets can often outperform methods based on traditional image models [Tod+16; Agu+17; The+17; Bur+12; Zha+17]. This success can largely be attributed to the ability of deep networks to represent realistic images when trained on large datasets. Examples include learned representations via autoencoders [Bis06] and generative adversarial models [Goo+14]. Almost exclusively, three common features of the recent success stories of using deep neural network for imaging related tasks are i) that the corresponding networks are over-parameterized (i.e., they have much more parameters than the dimension of the image that they represent or generate), ii) that the networks have a convolutional structure, and perhaps most importantly, iii) that the networks are trained on large datasets.

- The network is under-parameterized. Thus, the network maps a lower-dimensional space to a higher-dimensional space, similar to classical image representations such as sparse wavelet representations. This feature enables image compression by storing the coefficients of the network after its weights are optimized to fit a single image. In Section 2, we demonstrate that the compression is on-par with wavelet thresholding [Ant+92], a strong baseline that underlies JPEG-2000. An additional benefit of underparameterization is that it provides a barrier to overfitting, which enables regularization of inverse problems.

- The network itself acts as a natural data model. Not only does the network require no training (just as the DIP [Uly+18]); it also does not critically rely on regularization, for example by early stopping (in contrast to the DIP). The property of not involving learning has at least two benefits: The same network and code is usable for a number of applications, and the method is not sensitive to a potential misfit of training and test data.
Causal Inference

EXCERPTS FROM:
https://www.inference.vc/untitled/
- observational $p(y|x)$: What is the distribution of $Y$ given that I observe variable $X$ takes value $x$. This is what we usually estimate in supervised machine learning. It is a conditional distribution which can be calculated from $p(x, y, z, \ldots)$ as a ratio of two of its marginals. $p(y|x) = \frac{p(x,y)}{p(x)}$. We're all very familiar with this object and also know how to estimate this from data.

- interventional $p(y|do(x))$: What is the distribution of $Y$ if I were to set the value of $X$ to $x$. This describes the distribution of $Y$ I would observe if I intervened in the data generating process by artificially forcing the variable $X$ to take value $x$, but otherwise simulating the rest of the variables according to the original process that generated the data. (note that the data generating procedure is NOT the same as the joint distribution $p(x, y, z, \ldots)$ and this is an important detail).
training data $\sim$ trained ML model $q(y|x; \theta)$ $\sim$ observable joint $p(y|x)$
observational conditional

\[ p(y|x) \]

observable joint

\[ \sim \]

training data

intervention joint

\[ \approx \]

intervention conditional

\[ p(y|\text{do}(x)) \]

intervention model

\[ q(y|x; \theta) \]
A diagram illustrates the relationship between observational and interventional distributions in causal models. The process involves:

1. **Causal Model**: Represents the underlying causal system.
2. **Intervention**: Modifies the causal system by setting a variable to a specific value.
3. **Observational Joint**: The distribution of variables under observation.
4. **Estimable Formula**: \( E_{x' \sim p(x)} E_{z \sim p(z|x)} p(y|x', z) \)
5. **Conditionals, Marginals, and Expectations**: Adjustments for conditional probabilities and marginal distributions.
6. **Interventional Distribution**: The distribution of variables after intervention.
7. **Do Calculus**: Applies the do operator to simulate intervention.

The diagram shows how to move from observational to interventional distributions through a series of transformations and calculations.
\[
x = \text{randn()}
y = x + 1 + \sqrt{3}\times \text{randn()}
\]
\[
y = 1 + 2\times \text{randn()}
x = (y-1)/4 + \sqrt{3}\times \text{randn()}/2
\]
\[
z = \text{randn()}
y = z + 1 + \sqrt{3}\times \text{randn()}
x = z
\]
\[ x = \text{randn()} \]
\[ x = 3 \]
\[ y = x + 1 + \sqrt{3} \cdot \text{randn()} \]
\[ x = 3 \]

\[ y = 1 + 2 \cdot \text{randn()} \]
\[ x = 3 \]
\[ x = (y-1)/4 + \sqrt{3} \cdot \text{randn()}/2 \]
\[ x = 3 \]

\[ z = \text{randn()} \]
\[ x = 3 \]
\[ x = z \]
\[ x = 3 \]
\[ y = z + 1 + \sqrt{3} \cdot \text{randn()} \]
\[ x = 3 \]
CAUSAL MODEL

\[ P(u, v) \times P(x | u) \times P(a | x, v) \times P(b | x, v) \times P(q | x, a) \times P(s | a, q, b) \times P(c | a, q, b) \times P(y | s, u) \times P(z | v, c) \]

\[
x = f_1(u, \varepsilon_1) \quad \text{Query context } x \text{ from user intent } u.
\]
\[
a = f_2(x, v, \varepsilon_2) \quad \text{Eligible ads } (a_i) \text{ from query } x \text{ and inventory } v.
\]
\[
b = f_3(x, v, \varepsilon_3) \quad \text{Corresponding bids } (b_i).
\]
\[
q = f_4(x, a, \varepsilon_4) \quad \text{Scores } (q_{i,p}, R_p) \text{ from query } x \text{ and ads } a.
\]
\[
s = f_5(a, q, b, \varepsilon_5) \quad \text{Ad slate } s \text{ from eligible ads } a, \text{ scores } q \text{ and bids } b.
\]
\[
c = f_6(a, q, b, \varepsilon_6) \quad \text{Corresponding click prices } c.
\]
\[
y = f_7(s, u, \varepsilon_7) \quad \text{User clicks } y \text{ from ad slate } s \text{ and user intent } u.
\]
\[
z = f_8(y, c, \varepsilon_8) \quad \text{Revenue } z \text{ from clicks } y \text{ and prices } c.
\]
Inverse Problems
Inverse Problems

Quantity of interest  Measurement

Forward Model \( y = g(x) + n \)
Inverse Model \( \hat{x} = h(y) \)

Recurrent Inference Machines for Solving Inverse Problems

Patrick Putzky & Max Welling
Informatics Institute
University of Amsterdam
{pputzky,m.welling}@uva.nl

The Usual Approach

\[
L(X) = \log P_A(Y|X) + \log P_\theta(X) \\
Y = A \cdot X + \eta
\]

observations  generative model (known)  prior (learn)

\[
X_{t+1} = X_t + \alpha_t (\nabla_X \log P_A(Y|X_t) + \nabla_X \log P_\theta(X_t))
\]

advantage: model \( P(X) \) and optimization are separated.
disadvantage: accuracy suffers because model and optimization interact...

Learning Inference: Recurrent Inference Machine

- Abstract and parameterize computation graph into RNN
- Integrate prior \( P(X) \) in RNN
- Add memory state \( s \)
- Meta learn the parameters of the RNN

1 Introduction

Inverse Problems are a broad class of problems which can be encountered in all scientific disciplines, from the natural sciences to engineering. The task in inverse problems is to reconstruct a signal from observations that are subject to a known (or inferred) corruption process known as the forward model. In this work we will focus on linear measurement problems of the form

\[
y = Ax + n
\]

where \( y \) is a noisy measurement vector, \( x \) is the signal of interest, \( A \) is an \( m \times d \) corruption matrix, and \( n \) is an additive noise vector. If \( A \) is a wide matrix such that \( m \gg d \), this problem is typically ill-posed. Many signal reconstruction problems can be phrased in terms of the linear measurement problem such as image denoising, super-resolution, and deconvolution. The general form of \( A \) typically defines the problem class. If \( A \) is an identity matrix the problem is a denoising problem, while in tomography \( A \) represents a Fourier transform and a consecutive sub-sampling of the Fourier coefficients. In this paper we assume the forward model is known.

One way to approach inverse problems is by defining a likelihood and prior, and optimizing for the maximum a posteriori (MAP) solution:

\[
\max_x \log p(y|x) + \log p(x)
\]

Here, \( p(y|x) \) is a likelihood term representing the noisy forward model, and \( p(x) \) is a parametric prior over \( x \) which reduces the solution space for an otherwise ill-posed problem. In classical optimization frameworks there is a trade-off between expressiveness of the prior \( p(x) \) and optimization performance. While more expressive priors allow for better representation of the signal of interest, they will typically make optimization more difficult. In fact, only for a few trivial prior-likelihood pairs will inference remain convex. In practice one often has to resort to approximations of the objective and to approximate double-loop algorithms in order to allow for scalable inference.

Abstract and parameterize computation graph into RNN
Integrate prior \( P(X) \) in RNN
Add memory state \( s \)
Meta learn the parameters of the RNN

DATA-DRIVEN RECONSTRUCTION OF GRAVITATIONALLY LENSED GALAXIES USING RECURRENT INFERENCE MACHINES

WARREN R. MORNINGSTAR\textsuperscript{1}, LAURENCE PERREAU	extsc{L} LEVASSER\textsuperscript{3}, YASHAR D. HEZAVEH\textsuperscript{3}, ROGER BLANDFORD\textsuperscript{1}, PHIL MARSHALL\textsuperscript{1}, PATRICK PUTZKY\textsuperscript{4}, THOMAS D. RUETER\textsuperscript{2}, RISA WECHSLER\textsuperscript{1}, AND MAX WELLING\textsuperscript{1}

Draft version January 8, 2019

ABSTRACT

We present a machine learning method for the reconstruction of the undistorted images of background sources in strongly lensed systems. This method treats the source as a pixelated image and utilizes the Recurrent Inference Machine (RIM) to iteratively reconstruct the background source given a lens model. Our architecture learns to minimize the likelihood of the model parameters (source pixels) given the data using the physical forward model (ray tracing simulations) while implicitly learning the prior of the source structure from the training data. This results in better performance compared to linear inversion methods, where the prior information is limited to the 2-point covariance of the source pixels approximated with a Gaussian form, and often specified in a relatively arbitrary manner. We combine our source reconstruction network with a convolutional neural network that predicts the parameters of the mass distribution in the lensing galaxies directly from telescope images, allowing a fully automated reconstruction of the background source images and the foreground mass distribution.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure1.png}
\caption{A diagram of the structure of the reconstruction. A proposed image of the source (A) is lensed using the forward model (B, a ray-tracing simulation) to produce the lensed arcs (C, model data). The likelihood is computed by comparing the model with data. The derivative of the likelihood with respect to the values of the source pixels is calculated (D). The derivative of the likelihood and the current estimate (s_t) are given to the RIM (E) to produce a new estimate for the source (s_{t+1}).}
\end{figure}
Recurrent machines for likelihood-free inference

Arthur Pesah*
KTH Royal Institute of Technology
Stockholm, Sweden

Antoine Wehenkel*
University of Liège
Liège, Belgium

Gilles Louppe
University of Liège
Liège, Belgium

Abstract

Likelihood-free inference is concerned with the estimation of the parameters of a non-differentiable stochastic simulator that best reproduce real observations. In the absence of a likelihood function, most of the existing inference methods optimize the simulator parameters through a handcrafted iterative procedure that tries to make the simulated data more similar to the observations. In this work, we explore whether meta-learning can be used in the likelihood-free context, for learning automatically from data an iterative optimization procedure that would solve likelihood-free inference problems. We design a recurrent inference machine that learns a sequence of parameter updates leading to good parameter estimates, without ever specifying some explicit notion of divergence between the simulated data and the real data distributions. We demonstrate our approach on toy simulators, showing promising results both in terms of performance and robustness.