An Alternative to Monte Carlo Generators via Deep Generative Models
Replacing a standard Monte Carlo simulation.

HEPP2020, 29 -31 January 2020, University of Venda
Thabang D. Lebese

card: thabang@aims.ac.za
Overview

• Brief History
• Introduction
• GANs
• GANs architecture
• GAN applications in HEP
• Model training
• Results
• Conclusion & outlook
• Backup
• References
Brief History

• A branch of Deep Learning: first attempts in 1940s
• Pioneered by Ian Goodfellow. et al. in 2014
• Rapidly adopted
  • Goodfellow: @Google - 2015, @OpenAI - 2016, @Apple - 2019
  • Some applications
    • produce anime characters, convert scripts into animations, image character arithmetic,
    text-to-image synthesis, image-to-image translation, etc.
Introduction

• Two kinds of machine learning models:
  • generative and discriminative models

• The fundamental difference between two models lies in the underlying probability inference structure,
  • discriminative models learn $P(Y \mid X)$: the conditional relationship between the target variable $Y$ and features $X$ (limitations: Can’t model $P(X)$; thus, cannot sample from $P(X)$)
  • generative models aim for a complete probabilistic description of the dataset. Their goal is to develop the joint probability distribution $P(X, Y)$, either directly or by computing $P(Y \mid X)$ and $P(X)$ and then inferring the conditional probabilities required
  • to classify newer data. Can model $P(X)$, thus can generate new images, solve the curse of dimensionality, etc.

• Some types of Generative models:
  • Auto-Encoder (AE) = non-linear PCA
    • vanilla Feed Forward AE
    • convolutional AE
    • variation AE (Break through Research, uses the concept of Reconstruction)
    • Probability & Directed Probabilistic Graphical Model (DPGM)
  • sequence models
  • Generative Adversarial Networks (GANs)
    • Vanilla GAN
    • Wasserstein-GAN (wGAN), Conditional GAN (cGAN), CycleGAN, etc.
“Generative Adversarial Networks is the most interesting idea in the last 10 years in Machine Learning”

Yann LeCun, VP & Chief AI Scientist at Facebook & Professor at NYU (2016)
Generative Adversarial Nets (GANs)

- Idea: pit two deep subnets against each other
  - creating an adversarial “game” between Generator (G) & Discriminator (D)
  - G: generates fake samples from noise, tries to fool the D
  - D: tries to distinguish between real and fake samples
  - G & D train against each other repeatedly until we get a better Generator and Discriminator, i.e. until there’s some type of a Nash equilibrium (which may not happen)

- GANs use a minimax structure that uses the loss function constructed to minimize the Jensen-Shannon divergence between the distribution of the real data and the distribution of the generated data
noise (z) is an input into the Generator network (G) that outputs Fake data G(z) as input into the Discriminator network (D), then D outputs D(G(z)) that can be classified as Real = 1 or Fake = 0. Fine tune G and D weights during non-supervised training.
GANs in HEP

• Generative models can be viewed as:
  • a regression tasks that maps noise to structure, i.e. mimicking the Jacobian from a pre-defined distribution to target probability distribution

• shown great promise for accelerating simulations at the LHC
• useful for tasks such as sampling from the space of effective field theory models

• why GANs?
  • they can readily model asymmetric distributions and accommodate conditional features
  • they can be generic, accurate, fast and robust
Modeling

Features & preprocessing

Zero tail cut

le
p0
_ pt : min: 27.000242 | max: 1736.1593
le
p1
_ pt : min: 10.0000925 | max: 735.9781
TrackMET : min: 0.021143772 | max: 1328.2954
Mll : min: 10.001374 | max: 2178.0117
Ptll : min: 0.07439626 | max: 1728.6128
MT : min: 11.264636 | max: 2414.1794

5% tail cut

le
p0
_ pt : min: 27.000242 | max: 159.03975
le
p1
_ pt : min: 10.0000925 | max: 83.594124
TrackMET : min: 0.021143772 | max: 126.6691
Mll : min: 10.001374 | max: 257.86337
Ptll : min: 0.07439626 | max: 150.43474
MT : min: 11.264636 | max: 348.14563

Insights

- too much information loss
- evidence for possible convergence
Further modeling

- **no of features:** 6
- **applied cuts:** low = 0; high = 0.995
- **pretraining size:** 424872K
- **training size:** 800K
- **nb_epoch:** 800K
- **CPU training duration:** 09:26:54.29129

### Models

<table>
<thead>
<tr>
<th>Models</th>
<th>No. Epochs</th>
<th>Time</th>
<th>Can Discriminate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multilayer Perceptron (MLP)</td>
<td>1M</td>
<td>2:16:01.694724</td>
<td>False</td>
</tr>
<tr>
<td>Recurrent Neural Network (RNN)</td>
<td>150K</td>
<td>6:56:18.648813</td>
<td>False</td>
</tr>
<tr>
<td>Convolutional Neural Network (CNN)</td>
<td>500K</td>
<td>4:49:37.166094</td>
<td>True</td>
</tr>
</tbody>
</table>

Other architectures tried: fully connected, **RNNs, MLP** but **CNNs** yielded better performance and results, thanks to their superior ability to “learn” complex patterns.
GAN Performance w/ 5% tail cuts

ATLAS Simulation Work in Progress
GAN Performance w/ 0.5% tail cuts

ATLAS Simulation Work in Progress
Current status

- training is a **minmax** game, not a Minimization
- loss **stabilization & tuning**
- no trivial way to measure the agreement for choosing the best training epoch
- target goal is to end up with lowest $\chi^2$
- cross-entropy loss converges to $-\ln(0.5)=0.69314$
- SWAN CPUs, takes $\sim100k$ epochs / 1hr

Performance metrics

- "Nash Equilibrium" = 0.692646 $\pm 0.05005\%$
- "discrimination accuracy" = 0.5
Conclusion & outlook

• more generative model techniques are applied to HEP
  • Variational Autoencoders (VAE), Mixture Density Networks (MDN),
    Adversarial AE, GANs, etc.

• Generative nets to speed up generation of large Monte Carlo samples
  • use small “key” MC generated data with high accuracy, use GAN to increase
    statistics
  • quality of the inputs should be much closer to the truth (MC data)
  • GANs are not a minimization, picking up the best epoch is not trivial (e.g. lowest $\chi^2$)

• possible methods to consider:
  • direct training using TensorFlow on a GPU resourced environment
  • setup conditioning strategies for convergence speedup (TensorFlow)
  • use Auto-Encoder to handle arbitrary number of input variables
  • look into competing methods ($\beta$-VAE, Adversarial AE) ...
Backup
Software packages

• Keras v2.2.4
• TensorFlow v2.0.0
• scikit-learn 0.22.1,
• Pandas, other libraries
• Input scaled in the [0,1] range
Parameters

- Generators: 64 random number \(\sim U(0,1)\) \(\rightarrow\) 6 physics quantities:
- Loss functions:
  - Generator: mean square error (MSE)
  - Discriminator: binary cross-entropy
- Optimizer(s):
  - Stochastic Gradient Descent (SGD)
  - Learning rate = 10\(^{-3}\) \(\beta_1 = 0.5, \beta_2 = 0.5\) (slow gradient descent with momentum)
References


• Deep Learning books