GSOC REPORT

By Devesh Marwah
IIIT Hyderabad
India

Your improvements: Why and how they were implemented
Results: Show your results, at least by showing the output of the evaluation and analysis steps
Discussion

- Discuss the results
- Why your improvements work?
- What could be improved further
- What is better, a good overall evaluation of a good analysis result?
- Can you think of any fundamental flaws with Baler?
    - (We already know many!)

Initial results on all models

Before compression:

Mass : 174.94 +/- 0.12
Width : 5.52 +/- 0.13

	Mass	Width
george_SAE	170.53 +/- 0.28	13.94 +/- 0.33
george_SAE_Dropout_BN	168.81 +/- 0.45	-19.17 +/- 0.58
george_SAE_BN	164.56 +/- 0.29	-16.09 +/- 0.34
george_SAE_Dropout	224.89 +/- 16.94	-4.94 +/- 10.66

Code bug fixes in the repository:
Before discussing the improvements, I fixed the following few bugs in the repository:

1. **Error: mat1 and mat2 should have same dtype.**

- This error was encountered while running on other models and was a simple fix by adding

```
dtype=torch.float64,device=device
```

2. **Logical Error:** The forward function in `george_SAE_Dropout` was missing the self.encode(x) function which was added later.
3. The model was also setup on a remote cluster of GPU allotted by the university and tested there. The colab notebook here can also be used.

Dataset Used

The dataset used is the one provided by the mentor.

Basic Idea

Going through some of the GSOC projects from 2021, I came across an implementation of KL divergence in loss function for variational autoencoder here. Instead of implementing the same thing, I have experimented with three completely novel approaches, with two of them achieving significantly better results. These results can further be improved with layer finetuning, and adequate guidance from esteemed mentors of your organization.

Why did I go with the modification of KL divergence?

"It is apparent that VAE shows quite poor performance compared to the other two AEs. A reason for that could be the KL divergence term in its loss function which also showed to degrade the performance in the case of the SAE."

~ GSOC 2021 Project Report

- **Explanation of VAE**: VAEs are probabilistic models that learn a probability distribution across the latent space, whereas classic autoencoders simply learn a deterministic mapping from the input to the latent space. This is the primary distinction between VAEs and traditional autoencoders. In other words, by taking a sample from the previously learnt probability distribution, VAEs can produce new data points.

- **Why use a VAE?:** VAE can be especially powerful to learn a meaningful representation of high-dimensional data that captures its underlying structure.

This can be useful in scenarios where the goal is to extract features or patterns from the data. This can be seen in a lot of physics data and for high quantities of data there is an expectation of representations to act better.

- **Experimenting with loss functions**:

[Wasserstein distance]

- **The intuition behind the same**: Different from the regularizer employed by the Variational Auto-Encoder, WAE minimizes a penalized variant of the Wasserstein distance between the model distribution and the target distribution (VAE). The encoded training distribution is pushed to match the prior distribution by this regularizer.

- **How it is implemented**: It is implemented in the `vae_loss_function in the utils.py file.`

  Note that in order to replicate the results, you also need to return the additional encoded layer from the function.

Results:
After compression:
Mass : 169.16 +/- 0.32
Width : 8.49 +/- 0.35

As we can observe the results are quite poor but we will improve on the idea by using a simple intuitive modification of the same loss function!

[Sliced-Wasserstein Autoencoders]

Instead of using the KL divergence, I experimented with a different loss functions metric which can be also used in place to improve model's performance. The idea behind is to regularize the autoencoder loss with the sliced-Wasserstein distance between the distribution of the encoded training samples and a predefined samplable distribution (Generally Gaussian).

After compression:
Mass : 160.92 +/- 0.58

Width : 17.68 +/- 0.7

Understanding Adversarial AutoEncoders

While getting my hands dirty with the variational AutoEncoder loss function, I realized a seemingly basic problem which had far-reaching consequences.
The integral of the KL divergence term does not have a closed form analytical solution except for a handful of distributions.
Furthermore, it is not straightforward to use discrete distributions for the latent code z_dim. This is because backpropagation through discrete variables is generally not possible, making the model difficult to train efficiently.

In order to fix this, I experimented with the Adversarial Auto encoder (from here).

Traditional autoencoders frequently provide a low-dimensional representation that is unhelpful or meaningless for subsequent tasks. Adversarial training can be useful in this situation.

To distinguish between actual data and false data produced by the autoencoder, a second neural network called the **discriminator** must be trained using adversarial training. After that, the discriminator instructs the autoencoder to produce fake data that is identical to the genuine data.

The idea of adversarial training is to make the autoencoder develop a low-dimensional representation that not only captures the key characteristics of the data but also produces output that is realistic and resembles the real data.

How to run the code?: Change the model name to `adversial` in the `given_proj_config.py` file.

The code uses three major architectures implemented in models.py as Encoder, Decoder and Discriminative modules in the models.py folder.

Why it Works: Adversarial autoencoders avoid using the KL divergence altogether by using adversarial learning. In this architecture, a new network is trained to discriminatively predict whether a sample comes from the hidden code of the autoencoder or from the prior distribution. Here the distribution is gaussian. The loss of the encoder is now composed by the reconstruction loss plus the loss given by the discriminator network.

Results:

After compression:
Mass : 167.77 +/- 0.29
Width : -19.35 +/- 0.37

Future Improvements in The Project:

- A combination of Adversarial Networks along with Convolutions can be tested in order to capture the spatial relations in the data which are necessary to look over for physics data especially as it is possible for data to follow some hidden latent distributions.

- The current model of Baler does not make meaningful representations hence adversarial models can be used to improve the representations of the model.
- The adversarial model implemented here can be actually improved by layer finetuning. Multiple experiments for the same can be ran and need for a high end cluster is needed.

What is better, a good overall evaluation of a good analysis result?

Evaluation can be a better metric than analysis as it shows the nitty gritty of the performance. However analysis results can be used for crude approximation and get an eyeball view on the dataset.

Impact on Society:

Theoretical physics has a significant impact on society which is not visible for a normal man. Compression is a major issue and a lot of research has been behind this. The work presented here helps the researchers to step out from conventional models and present new work to help them explore new domains. Adversarial models have gained a lot of traction recently and have recently surpassed the state of the art models for various downstream tasks. This a step to work in that direction and improve the same.