Busting the hype around Generative models

AGENDA

- Generative Models in a nutshell

- Latent Dirichlet Algorithm
  - Case for topic modelling
  - Mixture models
  - Simulation of mixture models
  - Simulation of latent Dirichlet allocation

- Beyond classic Generative Adversarial Networks
  - Alternate loss function to improve training time of GANs.

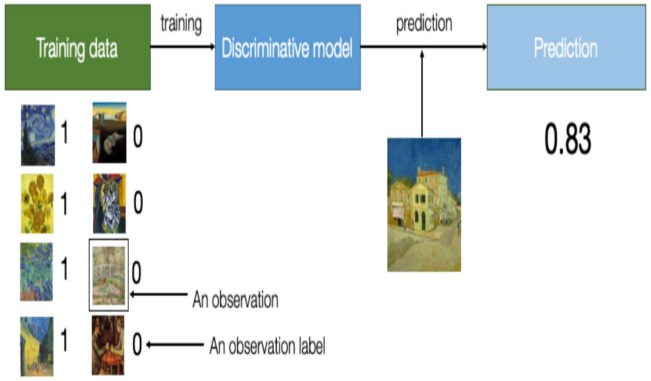
- After thoughts
  - Latest advancements in NLP leveraging GANs
  - Can LDA and GANs complement each other?

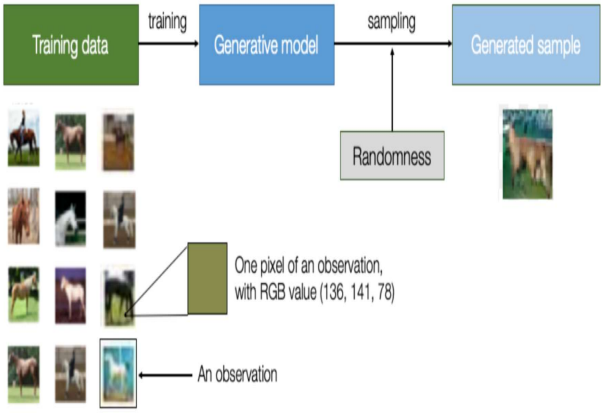
Generative Models in a nutshell

- Generative models describe how data is generated in terms of a probabilistic model.
    - By sampling from this model, we are able to generate new data.

- Generative modelling techniques usually performed with an unlabelled dataset.
    - It can **also** be applied to a labelled dataset to learn how to *generate observations from each distinct class*

Generative models vs Discriminative models





Learn $p(y|x)$ directly

Model $p(y)$, $p(x|y)$ first,
Then derive the posterior distribution:

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$

- Discriminative models are excellent tools in classifying data e.g. differentiating malignant cells from non-malignant cell.
- Not all problems however are classification problems. There are use cases that require deeper understanding of data and the factors that cause variations.
    - Unlabelled datasets for example cannot be classified using Generative techniques since there is no labelled data to start with.
    - Discriminative modelling is synonymous with *supervised learning* -- learning a function that maps an input to an output using a *labelled dataset*.

Latent Dirichlet Algorithm

- Latent Dirichlet Allocation and related topic models are often presented in the form of complicated equations like

$$
\begin{aligned}
w_i|z_i,\ \phi^{(z_i)} &\sim \text{Discrete}(\phi^{(z_i)}) \\
\phi &\sim \text{Dirichlet}(\beta) \\
z_i|\theta^{(d_i)} &\sim \text{Discrete}(\theta^{(d_i)}) \\
\theta &\sim \text{Dirichlet}(\alpha)
\end{aligned}
$$

  - The equations are not only complicated but also share very less insights on how to apply LDA to business use cases.
    - *for if the use case involves, discovering all Italian restaurants in a city or finding the ingredients of a Mexican cuisine even though LDA is an excellent choice of algorithm the equations fails to capture that.*

  - An effective yet simple approach to understand LDA would be to treat it as a Generative model through probabilistic simulation in simple Jupyter Notebook.
    - *Simulation will help to understand the model assumptions and limitations and more effectively use black box LDA implementations.*

Simulation over Iris dataset with fixed mean and standard deviation

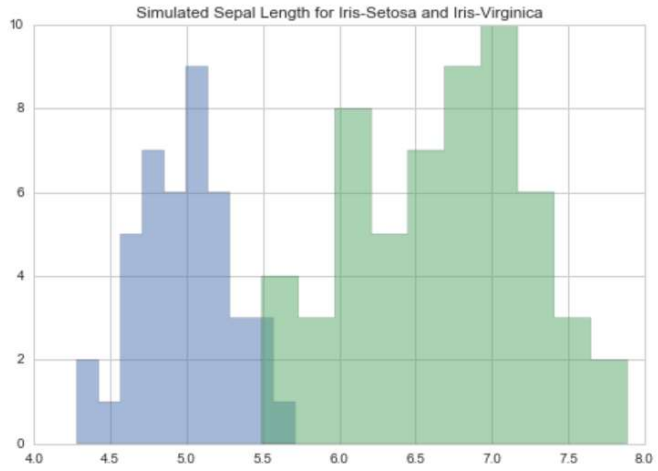
- We are running 100 iterations and randomly generating an iris type per iteration and then using normal distribution to calculate the sepal length

```
# Fixed model parameters
mixture_proportion = [0.50, 0.50]
mean               = [5.01, 6.59]
std                = [0.35, 0.64]

data = []
for _ in range(100):
    iris_type = random.choice(2, p=mixture_proportion)
    sepal_length = norm(loc=mean[iris_type],
                        scale=std[iris_type]
                       ).rvs()

    data.append((iris_type, sepal_length))
```

```
plt.hist([sepal_length for iris_type, sepal_length
          in data if iris_type==0], **settings)
plt.hist([sepal_length for iris_type, sepal_length
          in data if iris_type==1], **settings)
_=plt.title('Simulated Sepal Length for Iris-Setosa and Iris-Virginica')
```



- What if instead of taking the fixed proportion of the flower types (0.5, 0.5 in our example), we attempt to generate the sample via a distribution.

```
# Distribution on parameters
mixture_proportion = dirichlet(alpha=[1, 1]).rvs()[0]
mean               = norm(loc=6, scale=1).rvs(size=2)
std                = norm(loc=.5, scale=.2).rvs(size=2)

data = []
for z in range(1, 101):
    iris_type = random.choice(2, p=mixture_proportion)
    sepal_length = norm(loc=mean[iris_type],
            scale=std[iris_type]
        ).rvs()
    data.append((iris_type, sepal_length))
```

- *This distribution consists of continuous distribution over unit vectors.*
- *Each sample is often regarded as a probability distribution*
- *The relative number of datapoints is determined by the parameters of distribution.*

This in short represents the innocuous yet very effective Dirichlet distribution over discrete probability which attempts to model the underlying process that generates the data.

- which was represented by daunting mathematical equations as the start.

How is this different from the fixed distribution??

The data generated by the distribution is different in different draws as against the first case where we had the fixed sample size of (0.5, 0.5).

Business use case for LDA

- We can extend the same concept to find the latent parameters that are discussed in a review.
- We take YELP dataset and would apply LDA to find what exactly people are discussing when they visit restaurants in a given city/location.
    - This can be very handy in constructing NLP pipelines that involves Named Entity recognition (NER) and Named Entity Disambiguation (NED) and Named Entity Linking (NEL).

- Notebook is available on https://github.com/saurav-joshi/statistical-inferences/blob/master/yelp_data_challenge.ipynb
    - We will use to this to discuss in details how LDA can be used for

Beyond classic Generative Adversarial Networks

- Generative Adversarial Networks (GANs) have been an area of active research from past few years.

- Classic GANs consisting of an untrained Generator and untrained Discriminator are hard to train, some common training issues include
  1. **Difficult to find convergence**: The model parameters at time takes quite a long time to converge. This limits the scope of GANs.
  2. **Diminished gradient:** Generators at times learns too little or nothing due to the Diminished gradient problem.
  3. **Overfitting:** Unbalance between the Generator and Discriminator leads to overfitting of training data.
  4. **Hyper Sensitivity to Hyperparameter selection:** Batch size, **epoch, nature of the images--** low resolution, medium resolution, high resolution**--** can largely affect the training and performance of GANs.

- Generator and Discriminator in Vanilla GANs starts without any pretraining and hence it's extremely time consuming to arrive at the Nash equilibrium/steady state.

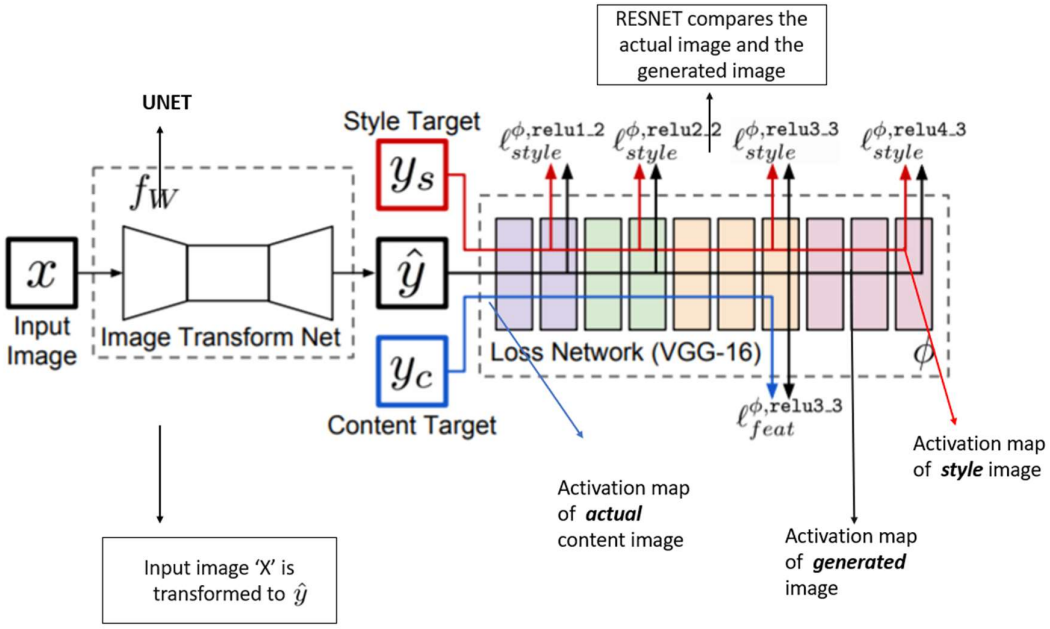
Alternate Loss function to expedite Nash equilibrium

- Inspired from paper https://arxiv.org/abs/1805.08318 , our Generator is a pretrained U-Net with spectral normalization and self-attention.
  - Pretrained Generator will drastically improve the convergence time.

- Training methodology is inspired by Progressive Growing of GANs https://arxiv.org/abs/1710.10196. A conscious effort is made to divide images into three categories name low-resolution images, medium-resolution images and high-resolution images.
  - The network is first trained on low res images and to learn the features of low-resolution images.
  - The trained network is then incrementally trained on medium-resolution images. The learning is then carried forward and the network is finally trained on high-resolution images
  - The training procedure of incrementally training based on image type, and not as one type fit all training, aptly concurs the name Progressive Growing of GANs

- Inspired from paper https://arxiv.org/pdf/1603.08155.pdf our Generator loss is the composition of two losses.

1. **Feature loss**: Feature loss measures the difference in the style and content of the generated images against the real image.
    a. We use UNET to transform the input image and then compare the loss with the real image.
2. **Discriminator loss**: Discriminator compares the activation map of generated and real images. A pretrained ImageNet network is used to calculate compare the activation maps of real and generated images.
    a. We use ResNet as the pretrained network for calculating Discriminator loss.



Hands on Exercise

- Our dataset consists of 200k+ celebrity images each with 40 attributes and annotations.
    - http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html has more details on the dataset.

- We will use TensorFlow 2.x alpha with GPU support.

- We will use BM. GPU 3.8 provided by Oracle cloud infrastructure with NVIDIA 8 Tesla V100. https://cloud.oracle.com/compute/gpu/features has more details on the infrastructure.