CS 615 – Deep Learning

Generative (Adversarial) Networks

Generative Models

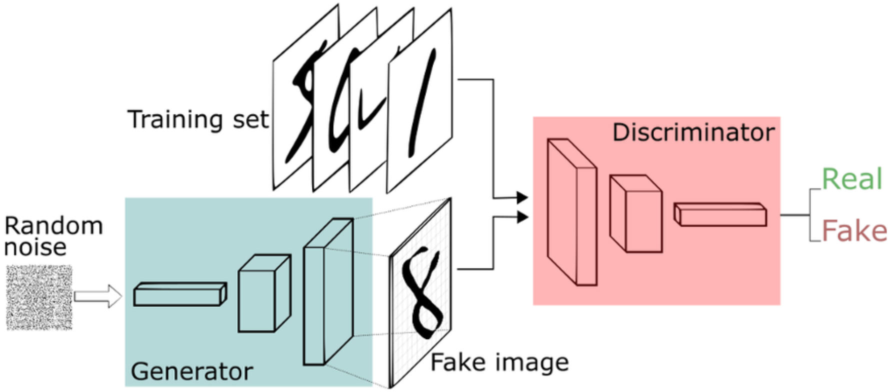
- With RNNs and LSTMs we talked about how a one-to-many system can be used as a *generative model*
  - Given an initial seed/input, x, generate outputs $y^{(1)}, y^{(2)}, \dots, y^{(T)}$
  - For example, generating text based on a seed input.
- Another approach to creating a model that can generate “realistic” output, is to create a *generative adversarial network (GAN).*

Generative Adversarial Networks (GAN)

- A Generative Adversarial Network (GAN) uses a game-theoretic approach to learn from a training distribution
- It is a framework proposed by Ian Goodfellow, Yoshua Bengio and others in 2014
- It is a hot topic of discussion and research these days because of its high potential

GAN

- Basic idea of a GAN:
  - It is a “game” between two players: a generator (G) and a discriminator (D)
  - The generator tries to fool the discriminator by generating realistic-looking data
  - The discriminator tries to distinguish between real (training set) and fake (generated) data



GAN

- The goal of the generator is to learn how to generate data that has a similar distribution to the real data x
- We will call $G(z)$ our *generator network*
  - It takes as its input a random feature vector, z, generated based on the training data's input distribution.
  - Its output will be some generated data $G(z)$
- Similarly, let $D(x)$ be our *discriminator network*
  - This takes as an input either a real observation, x, or one generated by the generator network, i.e. $G(z)$.
  - This networks returns the probability that it came from a real observation as opposed to having been generated by $G(z)$

GAN Objective Functions

- We train D and G simultaneously by playing a **two-player minimax game**
- Let's imagine that we feed our discriminative network several sets of real observations and fake/generated ones.
- Doing this over several "observations", since $D(x)$ returns a probability, we can define a likelihood objective function for our discriminant function:

$$J_d = \prod_{i=1}^{N} D(X_i)\left(1 - D\big(G(Z_i)\big)\right)$$

- Or we can take the log of this to get a log likelihood objective function:

$$J_d = \sum_{i=1}^{N} \left(\log\big(D(X_i)\big) + \log\big(1 - D(G(Z_i)\,)\big)\right)$$

GAN Objective Functions

$$J_d = \sum_{i=1}^{N} \left(\log\big(D(X_i)\big) + \log\big(1 - D(G(Z_i)\,)\big) \right)$$

- So the discriminator wants to *maximize this.*
- The generative network wants to trick the discriminator.
- So its objective function for the generative network could be:

$$J_g = \sum_{i=1}^{N} \left(\log\big(D(G(Z_i)\,)\big) \right)$$

Training GAN

- Just alternate between optimizing J_d and optimizing J_g using all the data (full batch)
  - Can be computationally prohibitive
  - May result in overfitting for finite datasets.
- So instead, in practice, we alternate between k steps of optimizing J_d with mini-batches and one step of optimizing J_g

Pseudocode

1. Obtain the distribution p_z based on training data.
2. Until termination criteria is met
    1. For k steps
        1. Create m (half of mini-batch) “fake” input vectors based on the distribution p_z.
        2. Grab m (other half of mini-batch) true input vectors from the training data.
        3. Update/train D using this mini-batch
    2. Generate m (half of mini-batch) “fake” input vectors based on distribution p_z.
    3. Grab m (other half of mini-batch) true input vectors from the training data.
    4. Update/train G using this mini-batch

Generated Samples

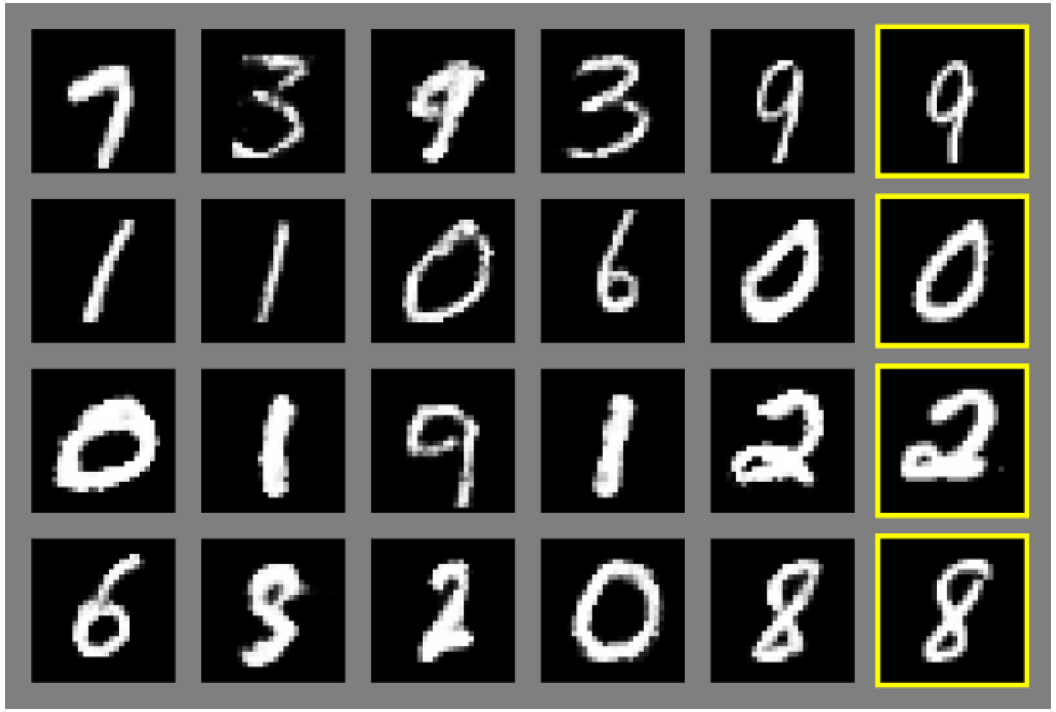


Fig1: MNIST dataset (Ian Goodfellow)

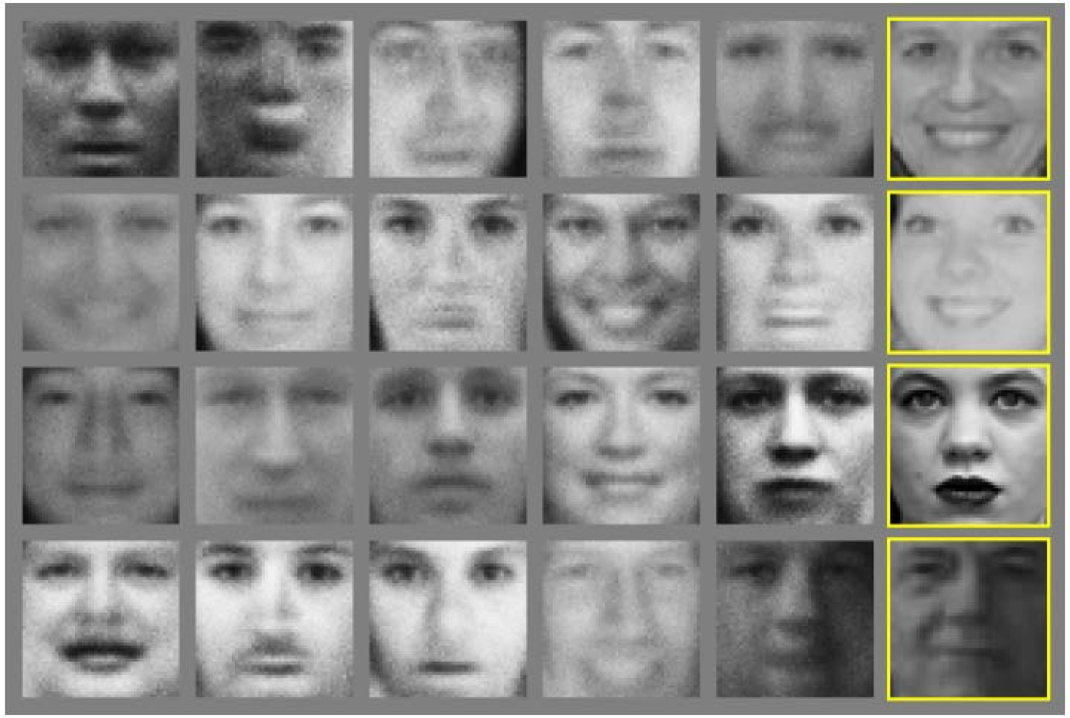


Fig2: TFD dataset (Ian Goodfellow)

The rightmost column shows the nearest training example of the neighboring sample

Bibliography

- Goodfellow, Ian, et al. "Generative adversarial nets." *Advances in neural information processing systems*. 2014.
- https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29