Advanced Machine Learning Subsidary Notes

Lecture 19: Wasserstein GANs

Adam Prügel-Bennett

February 9, 2024

1 Keywords

- GANs, Wasserstein distance, Duality, WGANs

2 Main Points

2.1 Generative Adversarial Networks GANs

- GANs are generative models
- They are often used for generating images or text
  - we will consider images just to be concrete but nothing really changes if we use text
- The task of the GAN is to generate images from the same distribution as those of a dataset $\mathcal{D}$
- GANs use two networks
  1. *A generator network*
     - It is fed a random vector $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0}, \mathbf{I})$
     - They are usually networks made with fully connected and deconvolution layers with weights $\boldsymbol{w}_G$
     - They generate an "image" $\hat{\boldsymbol{x}} = G(\boldsymbol{z}, \boldsymbol{w}_G)$
     - We train the weights to deceive the discriminator network
  2. *A descriminator network*
     - They receive either
       * a random sample from $\mathcal{D}$ or
       * an image $\hat{\boldsymbol{x}}$ generated by the generator seeded with a random vector $\boldsymbol{z}$
     - They are trained using backpropagation where the target output is
       * 1 for the real image from $\mathcal{D}$ or
       * 0 if the input is from the generator
     - They are usually CNN networks
- The generator weights are also learned by back-propagation through both the descriminator and generator networks where the target is for the descriminator for output 1
  - the descriminator weights aren't changed
  - this is opposite to the loss for the descriminator
  - it is fed the information about how to fool the descriminator (i.e. how to change the elements of $\hat{\boldsymbol{x}}$ to maximise the output of the descriminator)

    - Hopefully over time the generator produces image more like those from $\mathcal{D}$

- **Problems with GANs**
    - GANs are notoriously hard to train
    - The training of the descriminator and generator can become decoupled
    - For example, the descriminator can become so good that any local change of $\hat{\boldsymbol{x}}$ doesn't fool the descriminator
        * but this means there is no gradient to direct the learning of the generator
    - To overcome this we consider a very different approach

2.2 Wasserstein Distance

- The Big Picture
    - We consider minimising the distance between the distribution of images generated by the generator (that is, the distribution of $\hat{\boldsymbol{x}} = G(\boldsymbol{z}, \boldsymbol{w}_G)$ where $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$) and the distribution of real images (where we consider $\mathcal{D}$ to be a set of samples drawn from this distribution)
    - How do we measure distances between probability distributions?
    - One of the most common methods is to use the KL-divergence

$$\mathrm{KL}(p\|q) = \int p(\boldsymbol{x}) \log\left(\frac{p(\boldsymbol{x})}{q(\boldsymbol{y})}\right) \mathrm{d}\boldsymbol{x}$$

        * Relatively nice to compute
        * Not a true distance (but that doesn't bother us)
        * Unfortunately it can get very large even when the probability distributions are relatively close together

- **Earth-Moving Distance**
    - An very natural distance measure is the minimum distances you have to move the probability mass in one distribution $p(\boldsymbol{x})$ to make it identical to a second distribution $q(\boldsymbol{x})$
    - This is also known as the *Wasserstein* distance
    - Although conceptually straightforward it is a bit nasty to compute
    - **Optimal Transport Policy**
        * We start from a transport policy $\gamma(\boldsymbol{x}, \boldsymbol{y})$ that tells us how much probability mass (or density) we need to move from probability distribution p at point $\boldsymbol{x}$ to probability distribution q at point $\boldsymbol{y}$
        * As we start with a distribution $p(\boldsymbol{x})$ we need

$$\int \gamma(\boldsymbol{x}, \boldsymbol{y}) \, \mathrm{d}\boldsymbol{y} = p(\boldsymbol{x})$$

        * As we end with a distribution $q(\boldsymbol{x})$ we require

$$\int \gamma(\boldsymbol{x}, \boldsymbol{y}) \, \mathrm{d}\boldsymbol{x} = q(\boldsymbol{y})$$

        * Note that $\gamma(\boldsymbol{x}, \boldsymbol{y})$ looks like a joint probability distribution
            · It is non-negative
            · Integrating over $\boldsymbol{x}$ and $\boldsymbol{y}$ we get 1

    * We denote the set of probability distributions that satisfy these constraints $\Lambda(p,q)$
    * The cost of a particular transport policy is

$$C(\gamma) = \iint d(\boldsymbol{x},\boldsymbol{y})\,\gamma(\boldsymbol{x},\boldsymbol{y})\,\mathrm{d}\boldsymbol{x}\,\mathrm{d}\boldsymbol{y} = \mathbb{E}_\gamma[d(\boldsymbol{x},\boldsymbol{y})]$$

      since $\gamma(\boldsymbol{x},\boldsymbol{y})$ is the amount of probability mass we move and $d(\boldsymbol{x},\boldsymbol{y})$ is the distance we move it
    * The optimal transport policy is the distribution $\gamma \in \Lambda(p,q)$ with the minimum cost
    * The cost of the optimal transport policy is the Wasserstein distance

$$W(p,q) = \min_{\gamma\in\Lambda(p,q)} \mathbb{E}_\gamma[d(\boldsymbol{x},\boldsymbol{y})]$$

    * For high dimensional probability distributions finding the optimal transport policy using this definition is impractical
  - **Linear Programming**
    * Computing the Wasserstein distance is a linear programming problem
    * We want to choose $\gamma(\boldsymbol{x},\boldsymbol{y})$ to minimise a linear objective function $C(\gamma)$ subject to linear constraints
    * We can write this as a Lagrange problem

$$\mathcal{L} = \int d(\boldsymbol{x},\boldsymbol{y})\,\gamma(\boldsymbol{x},\boldsymbol{y})\,\mathrm{d}\boldsymbol{x}\,\mathrm{d}\boldsymbol{y} - \int \alpha(\boldsymbol{x})\left(\int \gamma(\boldsymbol{x},\boldsymbol{y})\,\mathrm{d}\boldsymbol{y} - p(\boldsymbol{x})\right)\mathrm{d}\boldsymbol{x} \\ - \int \beta(\boldsymbol{y})\left(\int \gamma(\boldsymbol{x},\boldsymbol{y})\,\mathrm{d}\boldsymbol{x} - q(\boldsymbol{y})\right)\mathrm{d}\boldsymbol{y}$$

      subject to $\gamma(\boldsymbol{x},\boldsymbol{y}) \geq 0$
      · $\alpha(\boldsymbol{x})$ and $\beta(\boldsymbol{y})$ are Lagrange multiplier functions
      · This looks strange because we are used to optimise vectors in Linear programming but here we optimise functions
      · We can discretise the function and we would get a vector
      · But functions form a vector space so we can define a linear programme for functions
    * **Dual Form**
      · We can rearrange the Lagrangian as

$$\mathcal{L} = \int \alpha(\boldsymbol{x})\,p(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} + \int \beta(\boldsymbol{y})\,q(\boldsymbol{y})\,\mathrm{d}\boldsymbol{y} - \int \gamma(\boldsymbol{x},\boldsymbol{y})\left(\alpha(\boldsymbol{x}) + \beta(\boldsymbol{y}) - d(\boldsymbol{x},\boldsymbol{y})\right)\mathrm{d}\boldsymbol{x}\,\mathrm{d}\boldsymbol{y}$$

      · Now we can interpret $\gamma(\boldsymbol{x},\boldsymbol{y})$ as a Lagrange multiplier function so that the dual problem is

$$\max_{\alpha(\boldsymbol{x}),\beta(\boldsymbol{x})} \int \alpha(\boldsymbol{x})\,p(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} + \int \beta(\boldsymbol{y})\,q(\boldsymbol{y})\,\mathrm{d}\boldsymbol{y}$$

        subject to

$$\alpha(\boldsymbol{x}) + \beta(\boldsymbol{y}) \leq d(\boldsymbol{x},\boldsymbol{y})$$

      · note this is an inequality constraint because $\gamma(\boldsymbol{x},\boldsymbol{y}) \geq 0$
      · But this has to be true when $\boldsymbol{x} = \boldsymbol{y}$ so

$$\alpha(\boldsymbol{x}) + \beta(\boldsymbol{x}) \leq d(\boldsymbol{x},\boldsymbol{x}) = 0$$

      · Thus $\beta(\boldsymbol{x}) = -\alpha(\boldsymbol{x}) - \epsilon(\boldsymbol{x})$ where $\epsilon(\boldsymbol{x}) \geq 0$
      · Our objective function becomes

$$\int \alpha(\boldsymbol{x})\,p(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} + \int \beta(\boldsymbol{y})\,q(\boldsymbol{y})\,\mathrm{d}\boldsymbol{y} = \int \alpha(\boldsymbol{x})\,\left(p(\boldsymbol{x}) - q(\boldsymbol{x})\right)\mathrm{d}\boldsymbol{x} - \int q(\boldsymbol{x})\,\epsilon(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}$$

- But this is clearly maximised when $\epsilon(\boldsymbol{x}) = 0$ therefore $\beta(\boldsymbol{x}) = -\alpha(\boldsymbol{x})$
- The problem simplifies to

$$\max_{\alpha(\boldsymbol{x})} \int \alpha(\boldsymbol{x})\, p(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x} - \int \alpha(\boldsymbol{y})\, q(\boldsymbol{y})\, \mathrm{d}\boldsymbol{y} = \max_{\alpha(\boldsymbol{x})} \left(\mathbb{E}_p[\alpha(\boldsymbol{x})] - \mathbb{E}_q[\alpha(\boldsymbol{x})] \right)$$

subject to

$$\alpha(\boldsymbol{x}) - \alpha(\boldsymbol{y}) \leq d(\boldsymbol{x}, \boldsymbol{y})$$

- functions $\alpha(\boldsymbol{x})$ that satisfy this constraint are known as *Lipschitz-1 functions*
- An equivalent condition is that

$$\|\boldsymbol{\nabla}_{\boldsymbol{x}} \alpha(\boldsymbol{x})\| \leq 1$$

- this is a continuity condition saying the output has to change slowly as we change the input

2.3 Wasserstein GANs

- In our Wasserstein GAN we train a generator to minimise the Wasserstein distance between the distribution of images from the generator and the true distribution
- We use mini-batches to approximate the expectations

$$\mathbb{E}_p[\alpha(\boldsymbol{x})] \approx \frac{1}{|\mathcal{B}|} \sum_{\boldsymbol{x} \in \mathcal{B}} \alpha(\boldsymbol{x}), \qquad \mathbb{E}_q[\alpha(\boldsymbol{x})] \approx \frac{1}{n} \sum_{i=1}^{n} \alpha(G(\boldsymbol{z}_i, \boldsymbol{w}_G))$$

- We need to find the function $\alpha(\boldsymbol{x})$ that maximises the difference between these expectations
- We make $\alpha(\boldsymbol{x})$ a neural network called the *critic*
  - this plays the same role as the discriminator in a normal GAN
  - Again we make this a CNN
  - The difference is it has to be Lipschitz-1
  - This is difficult to achieve and is usually bodged (you can read the literature if you are interested)
- Wasserstein GANs claim to solve many of the problems of normal GANs
  - They are not perfect because they only approximate the Lipschitz-1
- They are for me one of the elegant solutions in machine learning of the last few years