Math 5001 Final Project

Jacob Hauck

December 9, 2022

1 Image Registration

A patient has an organ scanned. A few months later, the same patient has the same organ scanned. It is of medical importance to determine what has changed in the organ between the two scans, which were performed at different times, from different angles, and possibly using different modalities. This is a typical example of an *image registration* problem.

The images may differ (1) in geometry (for example, the shape of the object in the image) and (2) in visual features (for example, colors and textures). Imaging an object with the *same* camera from two *different* positions will capture different geometry. On the other hand, imaging an object with *different* cameras from the *same* position will capture the same geometry but possibly different visual features (if, say, one camera took black and white pictures, and the other, color pictures).

To make a reliable comparison of two different images of the same object, we need to "factor out" differences caused by the imaging process so that only differences in the underlying object remain. This is essentially the definition of image registration.

In medical imaging the goal is usually to factor out only geometric differences because differences in visual features may be of importance; some imaging modalities capture complementary visual features, so factoring out these differences might destroy important diagnostic information.

Many techniques focus on the simpler case in which visual features are the same (for example, registering two CT scans); in this case, the problem consists of identifying the geometrical transformation between the two images, which is often a matter of pattern matching. If the two images have very different visual features, then the problem is more challenging; it becomes necessary to rely on the specific kinds of features present in the distribution of images being studied. For example, an algorithm that could register MRI and X-ray CT images might not be able to register ultrasound and PET images without some kind of modification to address the different kinds of features produced by these modalities.

1.1 Definitions and terms

Before we can properly define image registration we need to define an image. There are two perspectives on this: the idealized notion of an image as a function on $[0,1]^n$ (a "continuous" image), and the notion of an image as a discrete set of samples of a continuous image.

Definition 1. *Let $n, c \in \mathbb{N}$ and $s \in \mathbb{N}^n$.*

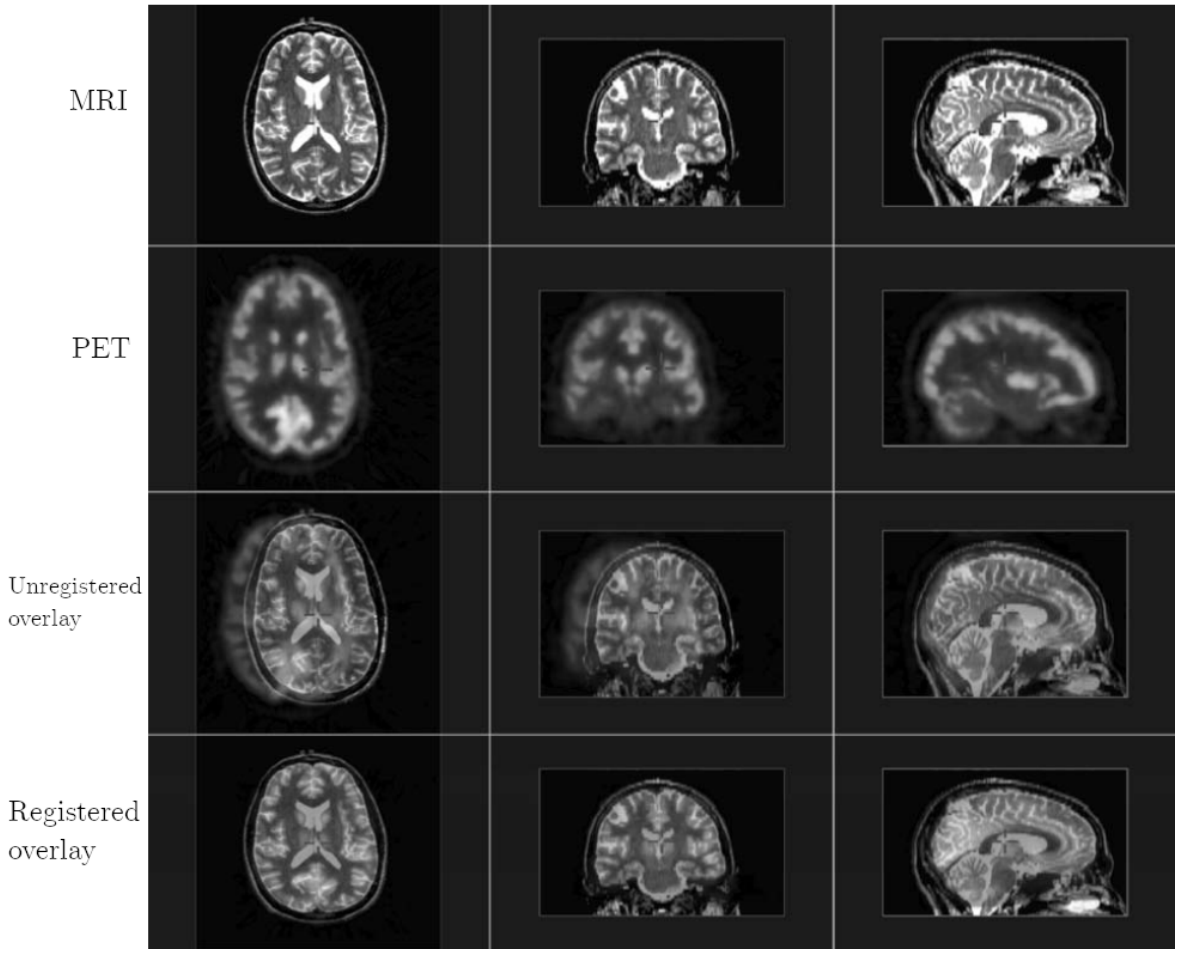

Figure 1: A comparison of MRI and PET images of the same person's brain. The bottom two rows show overlays of the images before and after registration. From Goshtasby's book on image registration [4].

- *An n-**dimensional continuous image** with c **channels** is a function $f : [0,1]^n \to \mathbb{R}^c$. Let*

$$C(n,c) = \{\text{all } n\text{-dimensional continuous images with } c \text{ channels}\} \tag{1}$$

- *An n-**dimensional discrete image** with c **channels** and **shape** s is a function $F : \mathbb{N}_0^s \to \mathbb{R}^c$, where*

$$\mathbb{N}_0^s := \{i \in (\mathbb{N} \cup \{0\})^n \mid i < s\}$$

Note that the $<$ above is taken in the sense of multi-index $<$. Let

$$D(n,c;s) = \{\text{all } n\text{-dimensional discrete images with } c \text{ channels and shape } s\} \tag{2}$$

We will use lower case letters for continuous images (f) and parentheses to get their values ($f(x)$). Likewise, we will use upper case letters for discrete images (F) and square brackets to get their values ($F[i]$).

Continuous images are easier to work with theoretically, but discrete images are a practical necessity. Therefore, we would like to relate the two. We do so by the following mapping.

Definition 2. *Let $s \in \mathbb{N}^n$, and define the **sampling grid** function $\mathscr{G}_s : \mathbb{N}_0^s \to [0,1]^n$ by*

$$\mathscr{G}_s(i) = \left(\frac{i_1 + \frac{1}{2}}{s_1}, \frac{i_2 + \frac{1}{2}}{s_2}, \ldots, \frac{i_n + \frac{1}{2}}{s_n}\right) \tag{3}$$

With this definition, we can immediately interpret the samples of an image $f \in C(n,c)$ as a discrete image $F \in D(n,c;s)$ with this definition

$$F[i] = f(\mathscr{G}_s(i)) \tag{4}$$

We would like to do this in the opposite direction as well, that is, interpret a discrete image as a continuous image, but there are many ways to do this because of the "gaps" in discrete images. In other words, we can't invert (4) because the grid sampling function is not onto. To address this issue, we introduce the notion of *interpolation*.

Definition 3. *Given an **interpolation kernel** function $\mathcal{K} : \mathbb{N}_0^s \times [0,1]^n \to \mathbb{R}$, the **interpolation** (by $\mathcal{K}$) of a discrete image F with shape s is the image f defined by*

$$f(x) = \sum_{i \in \mathbb{N}_0^s} F[i]\mathcal{K}(i,x) \tag{5}$$

We will use parentheses on an image to get the value of its interpolation; thus, the interpolation of $F \in D(n,c;s)$ by a kernel $\mathcal{K}$ at a point $x \in [0,1]^n$ is written $F_{\mathcal{K}}(x)$ or simply $F(x)$, if the specific kernel is understood or arbitrary. Conversely, we can write $f_s[i]$ or $f[i]$ to denote $f(\mathcal{G}_s(i))$.

Given an image $f \in C(n,c)$ of an object, suppose that we can assign a physical *reference point* (located in the object) to every $x \in [0,1]^n$, the domain of the image. Call this mapping the *reference map*, and denote it by $\mathcal{R}_f$.

Suppose we have another image $g \in C(n,c)$ of the same object. Then we get another reference map $\mathcal{R}_g$. This induces a relation on $[0,1]^n$ that relates points with the same reference point under $\mathcal{R}_f$ and $\mathcal{R}_g$. Let's assume that this correspondence can be written in terms of a transformation $\mathcal{T} : [0,1]^n \to [0,1]^n$:

$$\mathcal{R}_f(x) = \mathcal{R}_g(\mathcal{T}(x))$$

If we can find $\mathcal{T}$, then we can compute $g \circ \mathcal{T}$, which is "aligned" with f in the sense that $\mathcal{R}_{g\circ\mathcal{T}} = \mathcal{R}_g \circ \mathcal{T} = \mathcal{R}_f$. This is what is meant by "factoring out" geometrical differences in the images. The critical assumption that $\mathcal{R}_f$ and $\mathcal{R}_g$ are related by $\mathcal{T}$ makes the problem of image registration possible to solve (otherwise, we would have to deal with the intractable functions $\mathcal{R}_f$ and $\mathcal{R}_g$). Let's now define image registration properly.

Definition 4. *Let $f, g \in C(n, c)$ be images of the same object with reference maps $\mathcal{R}_f$ and $\mathcal{R}_g$. Suppose there is a transformation $\mathcal{T} : [0,1]^n \to [0,1]^n$ such that*

$$\mathcal{R}_f = \mathcal{R}_g \circ \mathcal{T} \tag{6}$$

*Then f is called the **fixed (or reference) image**, and g is called the **moving (or sensed) image**, and $\mathcal{T}$ is called the **transformation fuction**. The **image registration problem** is to determine $\mathcal{T}$ given f and g.*

*The **discrete image registration** problem is to determine the transformation function given only the discrete samples of f and g, that is, discrete images $F, G \in D(n, c; s)$ satisfying $F[i] = f[i]$ and $G[j] = g[j]$ for all $i, j \in \mathbb{N}_0^s$.*

1.2 Traditional approaches to image registration

The classical approach to image registration seeks "matching features" in the fixed and moving image, then determines a transformation function that makes the matching features coincide [4].

For example, in the registration of satellite images it is possible to segment bodies of water using thresholding (because water typically appears darker than land). In this case one would use the segmented regions as the features. To find matching features, one might compute transformation-invariant moments of the regions and define matching regions to be those with sufficiently close moments. Finally, one would find a transformation function that minimizes the distance between matching regions in the fixed and moving images.

In more recent approaches [2], one defines a *cost function* $\mathcal{C}(\mathcal{T}; f, g)$ that measures how badly a transformation function $\mathcal{T}$ registers the moving image g to the fixed image f[1]. Then the registration problem can be rephrased as an optimization problem in which the minimizer of $\mathcal{C}$ is the transformation that solves the registration problem. That is,

$$\mathcal{T}^* = \operatorname*{argmin}_{\mathcal{T}\in\Phi} \mathcal{C}(\mathcal{T}; f, g) \tag{7}$$

where Φ is a family of candidate transformation functions, and $\mathcal{T}^*$ is the transformation that solves the registration problem. Consider the following example of a cost function

$$\mathcal{C}^2(\mathcal{T}; f, g) = \int_{[0,1]^n} \left| f(x) - g(\mathcal{T}(x)) \right|^2 \mathrm{d}x \tag{8}$$

This cost function just measures the L^2 distance between f and $g \circ \mathcal{T}$, which would only work well if the images differ only by a transformation – a more refined cost function would be necessary if the fixed and moving image have significantly different visual features.

[1] All considerations in this section apply to the discrete image registration problem as well (possibly requiring interpolation or discretization).

Let $f, g \in C(n, 1)$, and extend f and g to $\mathbb{R}^n$ by setting $f(x) = 0 = g(x)$ for $x \notin [0,1]^n$. Suppose that Φ is the set of affine transformations of $\mathbb{R}^n$, and let $\mathcal{T}(x) = Ax + b$, where $A = (A_{ij}) \in \mathbb{R}^{n \times n}$, and $b = (b_i) \in \mathbb{R}^n$. Then we can write the gradient of $\mathcal{C}^2$ with respect to $\mathcal{T}$ as

$$\frac{\partial \mathcal{C}^2}{\partial A_{ij}} = 2 \int_{[0,1]^n} \Big(f(x) - g(Ax + b) \Big) x_j \frac{\partial g}{\partial x_i}(Ax + b) \, \mathrm{d}x \tag{9}$$

$$\frac{\partial \mathcal{C}^2}{\partial b_i} = 2 \int_{[0,1]^n} \Big(f(x) - g(Ax + b) \Big) \frac{\partial g}{\partial x_i}(Ax + b) \, \mathrm{d}x \tag{10}$$

If we can compute the integrals approximately, then we can get the approximate gradient of $\mathcal{C}^2$ with respect to $\mathcal{T}$. This means that we can use gradient descent to generate approximations of the minimizer $\mathcal{T}^*$. This type of iterative optimization scheme is typical of traditional approaches to image registration.

Remark 1. *The feature-matching approach to image registration is a special case of the cost function approach, where the cost function $\mathcal{C}$ is given by*

$$\mathcal{C}(\mathcal{T}; f, g) = d(\mathcal{F}(f), \mathcal{F}(g \circ \mathcal{T})) \tag{11}$$

where $\mathcal{F}$ is a feature extraction function, and d is a distance in the space of features. In the satellite imagery example, the feature extractor is the threshold and region-selection process. The distance would be a sum of absolute differences of the invariant moments of corresponding regions.

2 Generative Adversarial Networks

In some problems an appropriate cost function may be difficult to find, especially when the fixed and moving images have very different visual features. Assuming that examples of properly registered images are available[2], one might hope to infer an appropriate cost function based on the given examples.

This situation can be made to fit within the *generative adversarial framework*, in which one finds

1. An approximation $\mathcal{G}$ of a function $\mathfrak{G}$ such that

   $$Y \sim \mathfrak{G}(X)$$

   for random variables X and Y

2. An approximation d of a distance $\mathfrak{d}\left(\mathbb{P}_Y, \mathbb{P}_{\mathcal{G}(X)}\right)$ between the distributions $\mathbb{P}_Y$ and $\mathbb{P}_{\mathcal{G}(X)}$ of Y and $\mathcal{G}(X)$.

In the context of image registration, we would choose $X = (f, g)$, the pair of the fixed and moving images, and $Y = (f, g, \mathcal{T})$, where $\mathcal{T}$ is the transformation function for f and g. Then $\mathcal{G}$ would take two unregistered images to a transformation function that approximately solves the registration problem, and the distance d would approximately measure the quality of the registration across the distribution of images. Note that we choose $Y = (f, g, \mathcal{T})$ instead of $Y = \mathcal{T}$ because we are interested in modeling the relationship between (f, g) and $\mathcal{T}$; if we chose $Y = \mathcal{T}$, we would only be modeling the distribution of $\mathcal{T}$ without capturing its relationship to (f, g).

[2]This is not an unreasonable assumption since it is desirable to have a set of examples of properly registered images for the statistical evaluation of registration techniques.

The distance d approximated by the generative adversarial approach plays a role similar to the cost function from above; however, the cost function is applied to *particular* fixed-moving image pairs whereas the the distance d is applied to the *distribution* of fixed-moving image pairs and the *distribution* of transformation functions. The generative adversarial approach takes advantage of this important difference to produce an approximate representation of the solution mapping $f, g \mapsto \mathcal{T}$. By contrast, in the cost function approach the minimization of the cost function is the approximation of the solution mapping.

The subtle but important benefit of this approach is that it removes the need for *paired* data. In other words, all you need to apply this technique is a sample of registered images and another sample of unregistered images. Because the model learns at the level of distributions, it is not necessary to have corresponding registered images for every sample of unregistered images.

It is worth looking at two realizations of the generative adversarial framework: first, the original Generative Adversarial Network (GAN) framework and second, the successful, modified version called Wasserstein GAN (WGAN).

2.1 The original GAN formulation

Suppose we want to approximate the distribution of a random variable Y. One way to do this is to approximate its probability density function; however, this is not feasible when the Y is high-dimensional[3]. Instead, we can model the distribution of Y by finding a function $\mathcal{G}$ such that $\mathcal{G}(X)$ has approximately the same distribution as Y for some random variable X of our choosing.

To this end, let's try to find a function $\mathcal{G}$ that minimizes some distance between the distribution $\mathbb{P}_Y$ of Y and the distribution $\mathbb{P}_{\mathcal{G}(X)}$ of $\mathcal{G}(X)$. The original GAN authors [3] choose the Jensen-Shannon divergence, which is defined in terms of the Kullback-Leibler divergence as follows.

Definition 5. *Let X and Y be random variables with probability density functions p_X and p_Y with respect to the measure μ.*

- *The **Kullback-Leibler divergence** (KL divergence) of the distributions $\mathbb{P}_X$ and $\mathbb{P}_Y$ of X and Y is*
$$\mathrm{KL}\left(\mathbb{P}_X \| \mathbb{P}_Y\right) = \int p_X(z) \log\left(\frac{p_X(z)}{p_Y(z)}\right) \, \mathrm{d}\mu(z) \tag{12}$$
- *Let $\mathbb{P}_m = \frac{\mathbb{P}_X + \mathbb{P}_Y}{2}$. Then the **Jensen-Shannon divergence**[4] (JS divergence) of the distributions $\mathbb{P}_X$ and $\mathbb{P}_Y$ is*
$$\mathrm{JS}\left(\mathbb{P}_X, \mathbb{P}_Y\right) = \frac{1}{2}\mathrm{KL}(\mathbb{P}_X \| \mathbb{P}_m) + \frac{1}{2}\mathrm{KL}(\mathbb{P}_Y \| \mathbb{P}_m) \tag{13}$$

The authors of the GAN paper argue that an iterative algorithm that updates $\mathcal{G}$ so that $\mathrm{JS}\left(\mathbb{P}_Y, \mathbb{P}_{\mathcal{G}(X)}\right)$ decreases should cause $\mathbb{P}_{\mathcal{G}(X)}$ to converge to $\mathbb{P}_Y$. Thus, if $\mathcal{G} = \mathcal{G}_\theta$ is a neural network with parameters θ, then (assuming the network has enough parameters) using gradient descent to update the paremeters of $\mathcal{G}_\theta$ should result in $\mathcal{G}_\theta(X)$ having roughly the same distribution as Y. In GAN parlance, the neural network $\mathcal{G}_\theta$ is known as the *generator*.

That is all good and well, but we want to imagine that we only have access to a random sample from the distribution of Y, which makes computing the JS divergence difficult[5]. To deal with this

[3] That is, nuch greater than 1 or 2 dimensions.

[4] The KL divergence may not be defined for some distributions, but the JS divergence is always defined because we can choose $\mu = \mathbb{P}_m$.

[5] If it wasn't difficult enough without knowing the density of Y.

difficulty, another neural network $\mathcal{D}_\varphi$ with parameters φ can be used to approximate $\mathrm{JS}\left(\mathbb{P}_X, \mathbb{P}_{\mathcal{G}_\theta(X)}\right)$ by minimizing the *GAN loss*, which is carefully designed to involve only expected values of functions of X and Y, which we can approximate by sample averages.

Definition 6. *The **GAN loss** between random variables X and Y of a generator $\mathcal{G}$ and a discriminator $\mathcal{D}$ is the function*

$$\mathscr{L}_{\mathrm{GAN}}(\mathcal{D};\mathcal{G}) = -\big(\mathbb{E}[\log(\mathcal{D}(Y))] + \mathbb{E}[\log(1-\mathcal{D}(\mathcal{G}(X)))]\big) \tag{14}$$

Theorem 1. *For a fixed generator $\mathcal{G}$, if $\mathcal{D}^*$ is the minimizer of $\mathscr{L}_{\mathrm{GAN}}(\,\cdot\,;\mathcal{G})$, then*

$$\mathrm{JS}\left(\mathbb{P}_Y, \mathbb{P}_{\mathcal{G}(X)}\right) = \log(2) - \frac{1}{2}\mathscr{L}_{\mathrm{GAN}}(\mathcal{D}^*;\mathcal{G})$$

Proof. Suppose that $\mathcal{D}^*$ minimizes $\mathscr{L}_{\mathrm{GAN}}(\,\cdot\,;\mathcal{G})$. Then it maximizes

$$-\mathscr{L}_{\mathrm{GAN}}(\mathcal{D};\mathcal{G}) = \int p_Y(y)\log(\mathcal{D}(y))\,\mathrm{d}\mu(y) + \int p_{\mathcal{G}(X)}(y)\log(1-\mathcal{D}(y))\,\mathrm{d}\mu(y) \tag{15}$$

$$= \int \left[p_Y(y)\log(\mathcal{D}(y)) + p_{\mathcal{G}(X)}(y)\log(1-\mathcal{D}(y))\right]\mathrm{d}\mu(y) \tag{16}$$

with respect to $\mathcal{D}$, where $p_{\mathcal{G}(X)}$ is the density of $\mathcal{G}(X)$ with respect to the measure μ. It is easy to verify that the function $h(t) = a\log(t) + b\log(1-t)$ achieves a unique, global maximum for $t = \frac{a}{a+b}$ for any $a \geq 0, b \geq 0$ such that $a+b \neq 0$. Then at each point y the integrand of (16) achieves a unique, global maximum when

$$\mathcal{D}(y) = \frac{p_Y(y)}{p_Y(y) + p_{\mathcal{G}(X)}(y)}, \qquad y \in \mathrm{supp}(p_Y) \cup \mathrm{supp}(p_{\mathcal{G}}(X))$$

(the value of $\mathcal{D}(y)$ outside of the supports of Y and $\mathcal{G}(X)$ does not affect the value of the integrand). This implies that (almost everywhere)

$$\mathcal{D}^*(y) = \frac{p_Y(y)}{p_Y(y) + p_{\mathcal{G}(X)}(y)}, \qquad y \in \mathrm{supp}(p_Y) \cup \mathrm{supp}(p_{\mathcal{G}}(X))$$

Therefore,

$$-\frac{1}{2}\mathscr{L}_{\mathrm{GAN}}(\mathcal{D}^*;\mathcal{G}) = \frac{1}{2}\int \left[p_Y(y)\log\left(\frac{p_Y(y)}{p_Y(y)+p_{\mathcal{G}(X)}(y)}\right) + p_{\mathcal{G}(X)}(y)\log\left(\frac{p_{\mathcal{G}(X)}(y)}{p_Y(y)+p_{\mathcal{G}(X)}(y)}\right)\right]\,\mathrm{d}\mu(y)$$

$$= \frac{1}{2}\mathrm{KL}\left(\mathbb{P}_Y \,\middle\|\, \frac{\mathbb{P}_Y + \mathbb{P}_{\mathcal{G}(X)}}{2}\right) + \frac{1}{2}\mathrm{KL}\left(\mathbb{P}_{\mathcal{G}(X)} \,\middle\|\, \frac{\mathbb{P}_Y + \mathbb{P}_{\mathcal{G}(X)}}{2}\right) + \log\left(\frac{1}{2}\right)$$
$$= \mathrm{JS}\left(\mathbb{P}_Y, \mathbb{P}_{\mathcal{G}(X)}\right) - \log(2)$$

□

Suppose we have a neural network $\mathcal{D}_\varphi$ that minimizes $\mathscr{L}_{\mathrm{GAN}}(\mathcal{D}_\varphi;\mathcal{G}_\theta)$. Then we can train the generator $\mathcal{G}_\theta$ to minimize $-\mathscr{L}_{\mathrm{GAN}}(\mathcal{D}_\varphi;\mathcal{G}_\theta)$, which by Theorem 1 is equivalent to minimizing $\mathrm{JS}\left(\mathbb{P}_Y, \mathbb{P}_{\mathcal{G}(X)}\right)$. In GAN parlance, $\mathcal{D}_\varphi$ is called the *discriminator*.

If we use gradient descent to minimize $-\mathscr{L}_{\mathrm{GAN}}(\mathcal{D}_\varphi;\mathcal{G}_\theta)$ with respect to θ, we notice that the first term $\mathbb{E}\left[\log(\mathcal{D}_\varphi(Y))\right]$ does not contribute to the gradient with respect to θ. Therefore, it is equivalent to minimize

Definition 7. *The **GAN generator loss** is*

$$\mathscr{L}_{\text{GAN,G}}(\mathcal{D};\mathcal{G}) = \mathbb{E}\left[\log(1-\mathcal{D}(\mathcal{G}(X)))\right] \tag{17}$$

There is a slight wrinkle at this point: the discriminator $\mathcal{D}_\varphi$ is only a good approximation of the JS divergence for a *particular* generator, so when we update the generator using gradient descent, we need to retrain the discriminator for the new and slightly different generator. Therefore, we need to use an alternating training algorithm: train the discriminator to optimality, update the generator, train the discriminator to optimality, update the generator, ...

In practice, it has been observed that early in training the GAN generator loss provides weak gradients. It is equivalent to minimize a different loss that provides larger gradients.

Definition 8. (The $-\log(D)$ trick) *The $-\log(D)$ **trick GAN generator loss** is the loss*

$$\mathscr{L}'_{\text{GAN,G}}(\mathcal{D};\mathcal{G}) = -\mathbb{E}\left[\log(\mathcal{D}(\mathcal{G}(X)))\right] \tag{18}$$

The equivalence of the $-\log(D)$ trick to the true GAN generator loss follows from the fact that both $-\log(x)$ and $\log(1-x)$ are decreasing towards a minimum as $x \in (0,1)$ approaches 1 (see Figure 2). The trick is effective because the value $x = \mathcal{D}(\mathcal{G}(X))$ produced by the discriminator early in training is usually very small because the generator is bad; when x is small, the gradient of $\log(1-x)$ is small, but the gradient of $-\log(x)$ is large.

Finally, if we use sample averages to approximate the various loss functions, we arrive at the GAN training algorithm (Algorithm 1).

Algorithm 1 Stochastic Gradient Descent GAN Training Algorithm

Initialize neural networks $\mathcal{D}_\varphi$ and $\mathcal{G}_\theta$ with parameters φ and θ
for number of training iterations **do**
  for number of discriminator training steps **do**
    $(x_i)_{i=1}^m \leftarrow$ random sample minibatch from the distribution of X
    $(y_i)_{i=1}^m \leftarrow$ random sample minibatch from the distribution of Y
    Compute the approximate GAN loss

$$\widehat{\mathscr{L}}_{\text{GAN}}(\mathcal{D}_\varphi;\mathcal{G}_\theta) = -\frac{1}{m}\sum_{i=1}^{m}\Big[\log(\mathcal{D}_\varphi(y_i)) + \log(1-\mathcal{D}_\varphi(\mathcal{G}_\theta(x_i)))\Big]$$

    Use your favorite gradient descent update rule to update φ with the gradient $\nabla_\varphi \widehat{\mathscr{L}}_{\text{GAN}}(\mathcal{D}_\varphi;\mathcal{G}_\theta)$
  end for
  $(x_i)_{i=1}^m \leftarrow$ random sample minibatch from the distribution of X
  Compute the approximate GAN generator loss (using the $-\log(D)$ trick)

$$\widehat{\mathscr{L}'}_{\text{GAN,G}}(\mathcal{D}_\varphi;\mathcal{G}_\theta) = -\frac{1}{m}\sum_{i=1}^{m}\log(\mathcal{D}_\varphi(\mathcal{G}_\theta(x_i)))$$

  Use your favorite gradient descent update rule to update θ with the gradient $\nabla_\theta \widehat{\mathscr{L}'}_{\text{GAN,G}}(\mathcal{D}_\varphi;\mathcal{G}_\theta)$
end for

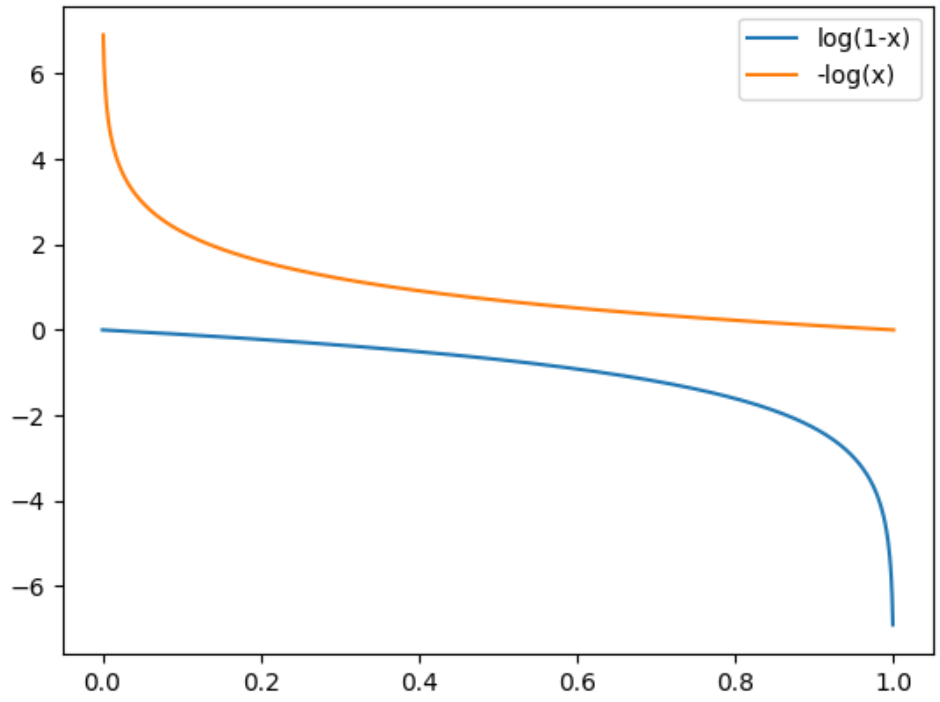

Figure 2: Comparison of $\log(1-x)$ and $-\log(x)$. Note that when x is close to 0 the function $\log(1-x)$ has a small slope, but $-\log(x)$ has a large slope. This helps improve training of the generator early.

Remark 2. *The GAN algorithm can also be thought of as a zero-sum, two-player game, in which the first player is the generator and the second player is the discriminator. The players compete with opposite goals: the discriminator attempts to discriminate between the true data distribution and the distribution of the generator, and the generator attempts to fool the discriminator. This is the origin of the term Generative Adversarial Network – the discriminator is the "adversary" of the generator in the game. The minimax equilibrium of the game corresponds to when the discriminator and the generator simultaneously minimize their respective losses.*

2.2 Wasserstein GAN

GANs can work very well, but they can also fail very well. For example, I know from experience that changing the learning rates of the discriminator and generator by even a little bit (10%, say) can make the difference between successful training and catastrophic failure. In general, the training process for GANs is known to be delicate and unstable.

Numerous techniques have been developed to address this extremely undesirable quality, for example, spectral normalization, label smoothing, noise terms, feature matching, and so on[6]. By far the most effective technique, Wasserstein GAN (WGAN) is based on a more careful mathematical analysis of the GAN training process.

What is happening when we train a GAN? Let's assume that the discriminator is always trained to near optimality. Then the training of the generator produces a sequence of parameters $\theta_1, \theta_2, \dots \theta_t, \dots$, which correspond to the sequence of distributions $\mathbb{P}_{\mathcal{G}_{\theta_t}(X)}$. We want that $\mathbb{P}_{\mathcal{G}_{\theta_t}(X)} \to \mathbb{P}_{\mathcal{G}_\theta(X)} \approx \mathbb{P}_Y$ in JS divergence as $\theta_t \to \theta$. In other words, we want the map $\theta \to \mathbb{P}_{\mathcal{G}_\theta(X)}$ to be continuous; in fact, if we want to compute the gradient of the JS distance with respect to θ, then we also want the JS divergence to be differentiable with respect to θ. Furthermore, we need the gradient with respect to θ to be nonzero in order for gradient descent to work.

This is where the problem arises. The JS divergence is actually a very strong distance (in the sense that it induces a strong topology). Roughly speaking, this makes it difficult for sequences to converge and difficult for functions to be continuous. This is thought to be the origin of the GAN training instability.

The best way to see the problem is through some examples. Let X and Y both be uniformly distributed on $[0, 1]$, and let $\mathcal{G}_\theta(x) = x + \theta$ for $\theta \in \mathbb{R}$. We can compute the JS divergence directly in this case. There are two possibilities to consider. The easier is when $|\theta| \geq 1$. In this case $\mathcal{G}_\theta(X)$ and Y have disjoint supports. Then the mixture distribution $\mathbb{P}_m = \frac{\mathbb{P}_Y + \mathbb{P}_{\mathcal{G}_\theta(X)}}{2}$ is uniform on the union $[0, 1] \cup [\theta, \theta + 1]$ of the supports. Therefore

$$\begin{aligned} \mathrm{KL}\left(\mathbb{P}_Y \| \mathbb{P}_m\right) &= \int_{[0,1]} p_Y(y) \log\left(\frac{p_Y(y)}{p_m(y)}\right) \, \mathrm{d}y \\ &= \int_{[0,1]} \log(2) \, \mathrm{d}y = \log(2) \end{aligned}$$

since the density p_m of the mixture distribution is constant and equal to $\frac{1}{2}$ on its support. A similar computation shows that $\mathrm{KL}\left(\mathbb{P}_{\mathcal{G}_\theta(X)} \middle\| \mathbb{P}_m\right) = \log(2)$. Thus, $\mathrm{JS}\left(\mathbb{P}_Y, \mathbb{P}_{\mathcal{G}_\theta(X)}\right) = \log(2)$. Now consider

[6] The $-\log(D)$ trick counts, as well.

the case when $|\theta| < 1$. Then the mixture distribution has two parts

$$p_m(y) = \begin{cases} \frac{1}{2} & y \in [0,1] \triangle [\theta, \theta+1] \\ 1 & y \in [0,1] \cap [\theta, \theta+1] \end{cases}$$

and we can write the KL divergence

$$\begin{aligned} \mathrm{KL}(\mathbb{P}_Y \| \mathbb{P}_m) &= \int_{[0,1]\setminus[\theta,\theta+1]} p_Y(y) \log\left(\frac{p_Y(y)}{p_m(y)}\right) \, \mathrm{d}y + \int_{[0,1]\cap[\theta,\theta+1]} p_Y(y) \log\left(\frac{p_Y(y)}{p_m(y)}\right) \, \mathrm{d}y \\ &= \int_{[0,1]\setminus[\theta,\theta+1]} \log(2) \, \mathrm{d}y = \log(2) \cdot \big|[0,1] \setminus [\theta, \theta+1]\big| \\ &= |\theta| \log(2) \end{aligned}$$

Similarly, we can write

$$\begin{aligned} \mathrm{KL}(\mathbb{P}_{\mathcal{G}_\theta(X)} \| \mathbb{P}_m) &= \int_{[\theta,\theta+1]\setminus[0,1]} p_{\mathcal{G}_\theta(X)}(y) \log\left(\frac{p_{\mathcal{G}_\theta(X)}(y)}{p_m(y)}\right) \, \mathrm{d}y + \int_{[0,1]\cap[\theta,\theta+1]} p_{\mathcal{G}_\theta(X)}(y) \log\left(\frac{p_{\mathcal{G}_\theta(X)}(y)}{p_m(y)}\right) \, \mathrm{d}y \\ &= \int_{[\theta,\theta+1]\setminus[0,1]} \log(2) \, \mathrm{d}y = \log(2) \cdot \big|[\theta, \theta+1] \setminus [0,1]\big| \\ &= |\theta| \log(2) \end{aligned}$$

Therefore $\mathrm{JS}\left(\mathbb{P}_Y, \mathbb{P}_{\mathcal{G}_\theta(X)}\right) = |\theta| \log(2)$ when $|\theta| < 1$. Altogether, then

$$\mathrm{JS}\left(\mathbb{P}_Y, \mathbb{P}_{\mathcal{G}_\theta(X)}\right) = \begin{cases} |\theta| \log(2) & |\theta| < 1 \\ \log(2) & |\theta| \geq 1 \end{cases} \tag{19}$$

We see from (19) that the JS divergence is continuous and almost everywhere differentiable with respect to θ, but this apparent victory is hollow; if $|\theta| > 1$, then the gradient is 0 (see Figure 3), so gradient descent will fail.

In the WGAN paper, the authors give an even more dramatic example. Let Z be uniformly distributed on $[0,1]$, and let $Y = X = (0, Z)$. Define $\mathcal{G}_\theta(x) = (\theta, x)$. Then

$$\mathrm{JS}\left(\mathbb{P}_Y, \mathbb{P}_{\mathcal{G}_\theta(X)}\right) = \begin{cases} \log(2) & \theta \neq 0 \\ 0 & \theta = 0 \end{cases} \tag{20}$$

Now the gradient vanishes everywhere, and gradient descent will always fail to make $\mathbb{P}_{\mathcal{G}_\theta(X)}$ converge to $\mathbb{P}_Y$, despite the fact that it is obvious by inspection what the optimal value of θ is. In fact, as $\theta_t \to 0$, the JS divergence $\not\to 0$, so the JS divergence is not even continuous at $\theta = 0$, the most important value of θ.

If the JS divergence is too strong, then what is a suitable, weaker distance? The authors of the WGAN paper choose the Wasserstein or earth mover (EM) distance[7].

[7] Theoretically, the reason that the EM distance is weak (and the motivation for choosing it in the first place) is the fact that it is a metrization of the weak* topology on the space of probability distributions on a compact probability space.

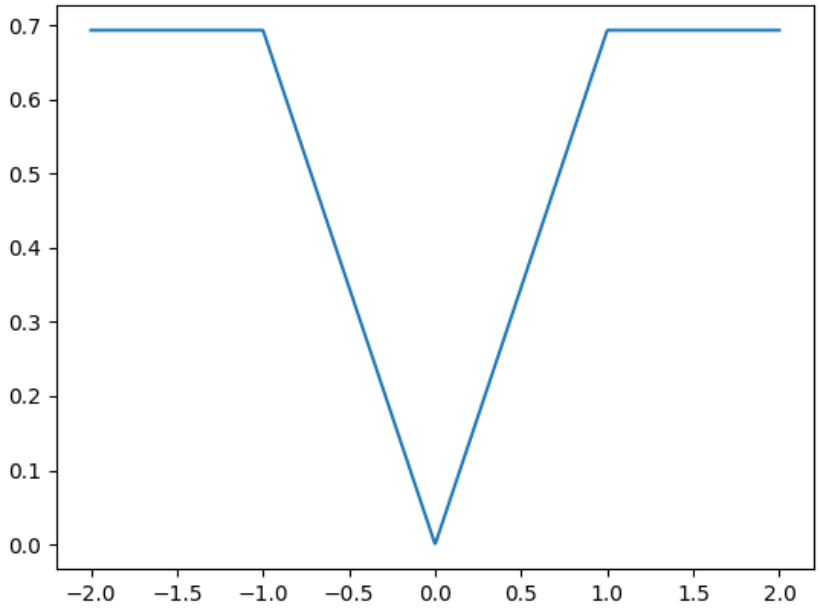

Figure 3: The JS divergence of $\mathbb{P}_Y$ and $\mathbb{P}_{\mathcal{G}_\theta(X)}$ as a function of θ. Note that the gradient descent will succeed in optimizing $\mathcal{G}_\theta$ when $|\theta| < 1$, but it will fail when $|\theta| > 1$.

Definition 9. *Given probability distributions $\mathbb{P}_X$ and $\mathbb{P}_Y$, the Wasserstein or earth mover (EM) distance is defined by*

$$W(\mathbb{P}_X, \mathbb{P}_Y) = \inf_{\gamma \in \Pi(\mathbb{P}_X, \mathbb{P}_Y)} \mathbb{E}_{(x,y)\sim\gamma}\big[|x - y|\big] \tag{21}$$

where $\Pi(\mathbb{P}_X, \mathbb{P}_Y)$ is the set of all joint distributions whose marginals are $\mathbb{P}_X$ and $\mathbb{P}_Y$.

In the example from the WGAN paper, the EM distance is just

$$W\left(\mathbb{P}_Y, \mathbb{P}_{\mathcal{G}_\theta(X)}\right) = |\theta| \tag{22}$$

Comparing to the JS divergence (20) we see that the EM distance is continuous and provides a useful gradient for all $\theta \in \mathbb{R}$, so we should expect gradient descent to work for training a generator. In fact, unlike the JS divergence, the EM distance is always continuous and almost everywhere differentiable:

Theorem 2. *[1] Let X and Y be random variables with distributions $\mathbb{P}_X$ and $\mathbb{P}_Y$, and let $\mathbb{P}_{\mathcal{G}_\theta(X)}$ be the distribution of $\mathcal{G}_\theta(X)$. Then*

1. *If $\mathcal{G}_\theta$ is continuous in θ, then $W\left(\mathbb{P}_Y, \mathbb{P}_{\mathcal{G}_\theta(X)}\right)$ is also continuous in θ.*
2. *If there exists $L(\theta, x)$ such that*

$$\|\mathcal{G}_\theta(x) - \mathcal{G}_{\theta'}(x')\| \leq L(\theta, x)(\|\theta - \theta'\| + \|x - x'\|) \quad \text{all } \theta', x' \tag{23}$$

$$\mathbb{E}[L(\theta, X)] < \infty \quad \text{all } \theta \tag{24}$$

then $W\left(\mathbb{P}_Y, \mathbb{P}_{\mathcal{G}_\theta(X)}\right)$ *is continuous everywhere and differentiable almost everywhere.*

3. Statements 1. and 2. are false for the JS divergence.

It is important to note that **neural networks** satisfy the hypotheses on $\mathcal{G}_\theta$ in Theorem 2, statements 1. and 2.

The following theorem completes the argument that the EM distance is the "right" distance for GAN training, at least in the sense that it is weaker than JS but still guarantess convergence in distribution.

Theorem 3. *[1] Let* $\mathbb{P}$ *be a probability distribution, and let* $(\mathbb{P}_n)_{n=1}^{\infty}$ *be a sequence of distributions. If* $\mathrm{JS}(\mathbb{P}, \mathbb{P}_n) \to 0$ *as* $n \to \infty$*, then* $W(\mathbb{P}, \mathbb{P}_n) \to 0$ *as* $n \to \infty$*. Furthermore,* $\mathbb{P}_n \overset{\text{dist}}{\to} \mathbb{P}$ *if and only if* $W(\mathbb{P}, \mathbb{P}_n) \to 0$*.*

So the EM distance is a better choice than the JS divergence. In order to use the EM distance, we need to devise a new GAN loss so that the discriminator approximates the EM distance instead of the JS divergence. Furthermore, we need this new loss to be expressible in terms of expected values so that we can approximate it with sample averages, as we did in the original GAN. This is actually fairly simple due to the following fact (a result of Kantorovich-Rubinstein duality).

Theorem 4. *[1] Let* $\mathbb{P}_X$ *and* $\mathbb{P}_Y$ *be the distributions of random variables* X *and* Y*. Let* $K > 0$*. Then the EM distance*

$$W(\mathbb{P}_Y, \mathbb{P}_X) = \frac{1}{K} \sup_{\|h\|_L \leq K} \big[\mathbb{E}[h(Y)] - \mathbb{E}[h(X)]\big] \tag{25}$$

where the supremum is taken over all real, K*-Lipschitz functions* h *on the probability space of* X *and* Y*.*

If we use a class of neural networks to approximate the set of all K-Lipschitz functions, then we can approximate the EM distance up to some unknown and irrelevant factor K by training the network to solve the maximization problem in (25). This is equivalent to minimizing the WGAN loss:

Definition 10. *Given a discriminator* $\mathcal{D}$ *and a generator* $\mathcal{G}$*, the* ***WGAN loss*** *is*

$$\mathscr{L}_{\text{WGAN}}(\mathcal{D}; \mathcal{G}) = \mathbb{E}\big[\mathcal{D}(\mathcal{G}(X))\big] - \mathbb{E}\big[\mathcal{D}(Y)\big] \tag{26}$$

Suppose we have a neural network $\mathcal{D}_\varphi$ that minimizes $\mathscr{L}_{\text{WGAN}}$; then, just like we did with the original GAN, we can train the generator $\mathcal{G}_\theta$ to minimize $-\mathscr{L}_{\text{WGAN}}(\mathcal{D}_\varphi; \mathcal{G}_\theta)$, which by Theorem 4 is equivalent to minimizing $W\left(\mathbb{P}_Y, \mathbb{P}_{\mathcal{G}_\theta(X)}\right)$.

The biggest difference from the original GAN algorithm is that we must somehow ensure that the network $\mathcal{D}_\varphi$ is K-Lipschitz for some K. The authors of the WGAN paper suggest weight clipping, that is, after each gradient descent update to $\varphi \in \mathbb{R}^\ell$, clip φ to the box $[-c, c]^\ell$, where c is some small, positive constant (a new hyperparameter in the WGAN framework).

If we use gradient descent to minimize $-\mathscr{L}_{\text{WGAN}}(\mathcal{D}_\varphi; \mathcal{G}_\theta)$ with respect to θ, we notice that the second term $-\mathbb{E}\big[\mathcal{D}(Y)\big]$ does not contribute to the gradient with respect to θ. Therefore, it is equivalent to minimize

Definition 11. *Given a discriminator* $\mathcal{D}$ *and a generator* $\mathcal{G}$*, the* ***WGAN generator loss*** *is*

$$\mathscr{L}_{\text{WGAN,G}}(\mathcal{D}, \mathcal{G}) = -\mathbb{E}\big[\mathcal{D}(\mathcal{G}(X))\big] \tag{27}$$

As in the original GAN, we can approximate the expected values by sample averages, which leads to the WGAN algorithm (Algorithm 2). The WGAN algorithm has enjoyed great empirical success, effectively curing the instability of the original GAN algorithm.

Algorithm 2 Stochastic Gradient Descent WGAN Training Algorithm

Initialize neural networks $\mathcal{D}_\varphi$ and $\mathcal{G}_\theta$ with parameters φ and θ
Fix a clipping parameter $c > 0$
for number of training iterations **do**
  for number of discriminator training steps **do**
    $(x_i)_{i=1}^m \leftarrow$ random sample minibatch from the distribution of X
    $(y_i)_{i=1}^m \leftarrow$ random sample minibatch from the distribution of Y
    Compute the approximate WGAN loss

$$\widehat{\mathscr{L}}_{\text{WGAN}}(\mathcal{D}_\varphi; \mathcal{G}_\theta) = \frac{1}{m} \sum_{i=1}^{m} \Big[\mathcal{D}_\varphi(\mathcal{G}_\theta(x_i)) - \mathcal{D}_\varphi(y_i) \Big]$$

    Use a gradient descent update rule* to update φ with the gradient $\nabla_\varphi \widehat{\mathscr{L}}_{\text{WGAN}}(\mathcal{D}_\varphi; \mathcal{G}_\theta)$
    Clip the parameter φ to the box $[-c, c]^\ell$
  end for
  $(x_i)_{i=1}^m \leftarrow$ random sample minibatch from the distribution of X
  Compute the approximate WGAN generator loss

$$\widehat{\mathscr{L}}_{\text{WGAN,G}}(\mathcal{D}_\varphi; \mathcal{G}_\theta) = -\frac{1}{m} \sum_{i=1}^{m} \mathcal{D}_\varphi(\mathcal{G}_\theta(x_i))$$

  Use a gradient descent update rule* to update θ with the gradient $\nabla_\theta \widehat{\mathscr{L}}_{\text{WGAN,G}}(\mathcal{D}_\varphi; \mathcal{G}_\theta)$
end for

* The gradient descent update rule should **not** use momentum. The authors of the original paper note that training can become unstable when using momentum-based rules.

3 Adversarial Image Registration

When the visual features of two images are very different, it is difficult to design a cost function to register them. Yan *et al.* [7] study one such case, involving the registration of MRI and ultrasound images. They attempt to overcome the differences in visual features by applying a WGAN in a manner similar to what was described at the beginning of section 2, but with one important difference (other than the insignificant change to discrete images).

As in section 2, let $X = (F, G)$ be a random pair of fixed and moving discrete images and $\mathcal{T}$ the transformation function that registers them. Instead of setting $Y = (F, G, \mathcal{T})$, Yan, *et al.* set $Y = (F, G \circ \mathcal{T})$, where $G \circ \mathcal{T}$ is defined by interpreting G as continuous through interpolation[8]. The benefit of this trick is that it allows the discriminator to classify whether images are registered instead of having to learn, independently of the generator, whether the transformation function $\mathcal{T}$ registers the images F, G. This is an easier task for convolutional networks, which excel at classification.

We will construct neural networks $\mathcal{D}_\varphi$ and $\mathcal{G}_\theta$ as the WGAN generator and discriminator. Since F and G are given, we will define the generator in two steps. The first step takes the images F and G and computes an approximate transformation function $\widehat{\mathcal{T}}$. The second step computes $\left(F, G \circ \widehat{\mathcal{T}}\right)$. If $\mathcal{G}_\theta$ is to be a neural network, then it cannot return anything but a vector, so in order to interpret its output as a transformation, we need some parameterization of a family of transformations. Let $\mathcal{T}(\lambda)$ denote the transformation corresponding to a parameter $\lambda \in \mathbb{R}^t$ (this notation emphasizes the dependence of $\mathcal{T}$ on λ). Then we want to make the output of the first step of $\mathcal{G}_\theta$ a vector in $\mathbb{R}^t$. We will define the family of transformations being used later.

The WGAN algorithm (Algorithm 2) takes care of the training of the neural networks $\mathcal{D}_\varphi$ and $\mathcal{G}_\theta$. All that is left is to specify the structure of these networks as they are defined by Yan, *et al.*

3.1 Convolutional architecture

The networks we will use for $\mathcal{D}_\varphi$ and $\mathcal{G}_\theta$ will be fairly simple convolutional neural networks. We now define what a convolutional network is, but only in the discrete case, as the network will actually be applied to discrete images. Instead of giving the usual discrete convolution definition, I will give a more general "machine learning" version, which includes a bias term as well as striding, dilation, and centering.

Definition 12. *Let $(k_H, k_W) \in \mathbb{N}^2$, $C_{\text{in}}, C_{\text{out}} \in \mathbb{N}$.*

- *If $W_c \in D\left(2, C_{\text{in}}; (k_H, k_W)\right)$ for $c = 1$ to C_{out}, and $b \in \mathbb{R}^{C_{\text{out}}}$, then $\Theta = \left((W_i)_{c=1}^{C_{\text{out}}}, b\right)$ is called a **convolution parameter** with **kernel size** $k = (k_H, k_W)$, **input channels** C_{in}, and **output channels** C_{out}. The images $W = (W_c)_{c=1}^{C_{\text{out}}}$ are called the **weights**, and the vector b is called the **bias**.*
- *The **discrete 2D convolution** with parameter Θ, **stride** $\sigma \in \mathbb{N}^2$, and **dilation** $d \in \mathbb{N}^2$ **centered at** $o \in \mathbb{N}_0^k$ is the function conv_Θ taking a discrete image $F \in D\left(2, C_{\text{in}}; s\right)$ to the discrete image $G \in D\left(2, C_{\text{out}}; s'\right)$ defined by*

$$G[i]_c = b_c + \sum_{j \in \mathbb{N}_0^k} W_c[j] \cdot F[\sigma i - d(j - o)] \qquad i \in \mathbb{N}_0^{s'},\ c = 1 \text{ to } C_{\text{out}} \tag{28}$$

[8] More on this in 3.2.

where we set $F[\ell] := 0$ *when* $\ell \notin \mathbb{N}_0^s$*, and* $s' = \left(\left\lfloor \frac{s_1}{\sigma_1} \right\rfloor, \left\lfloor \frac{s_2}{\sigma_2} \right\rfloor\right)$*. The symbol* σi *means* $(\sigma_1 i_1, \sigma_2 i_2)$*; a similar definition holds for* $d(j-o)$*. We will always choose the center* $o = \left(\left\lfloor \frac{k_H}{2} \right\rfloor, \left\lfloor \frac{k_W}{2} \right\rfloor\right)$*.*

Intuitively, one can think of a convolution as sliding the weights W over the input image F. Stride determines how "quickly" W slides over F; if $\sigma = (2,2)$, then the weights slide two pixels at a time, which should result in an output image that has half the resolution of the input. The dilation effectively "spreads out" the weights as they slide; if $d = (2,2)$, then the weights are spaced by two pixels as they slide instead of one[9] (see this nice animated visualization[10]).

Definition 13. *Define the* ***activation functions*** *(which map* $\mathbb{R} \to \mathbb{R}$*).*

1. *The sigmoid function* $\mathfrak{s}$ *defined by*
$$\mathfrak{s}(x) = \frac{1}{1+e^{-x}} \tag{29}$$
2. *The ReLU function* $\mathfrak{r}$ *defined by*
$$\mathfrak{r}(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{30}$$

We will apply these functions ***componentwise*** *to a discrete image* F *in the sense that* $\mathfrak{s}(F)$ *is the discrete image defined by* $\mathfrak{s}(F)[i] = \mathfrak{s}(F[i])$ *for all indices* i *into* F*.*

It has been found empirically that *residual learning* can improve the performance of deep convolutional neural networks [5]. The idea behind residual learning is to reparameterize the operations of the network as perturbations of the identity[11]. To this end, we define the *residual convolution block*.

Definition 14. *Given convolution parameters* $\Theta_1, \Theta_2, \ldots, \Theta_L$ *and an activation function* ρ*, the* ***2D residual convolution block*** *is the function* $\mathrm{resconv}_\rho(\Theta_1, \Theta_2, \ldots, \Theta_L)$*, which acts on an image* $F \in D(2, C_{\mathrm{in}}; s)$ *by*

$$\mathrm{resconv}_\rho(\Theta_1, \Theta_2, \ldots, \Theta_L)(F) = \left[\mathrm{conv}_{\Theta_1} \circ \rho \circ \mathrm{conv}_{\Theta_2} \circ \rho \circ \cdots \circ \mathrm{conv}_{\Theta_L}\right](F) + F \tag{31}$$

Now we are ready to define the networks $\mathcal{D}_\varphi$ and $\mathcal{G}_\theta$. Both networks will involve 7 convolutions. Let $\Theta_{\mathcal{D}} = \left(\Theta_{\mathcal{D}}^\ell\right)_{\ell=1}^7$ and $\Theta_{\mathcal{G}} = \left(\Theta_{\mathcal{G}}^\ell\right)_{\ell=1}^7$ be the parameters of these convolutions for the discriminator and generator networks. The kernel sizes, channels, strides and dilations of these convolutions are summarized in Table 1. Note that both the discriminator and generator convolutions use the same architecture parameters (kernel size, channels, stride, and dilation).

Then we define the convolutional subnetwork

$$C_{\mathcal{X}}(F,G) = \left[\mathfrak{r} \circ \mathrm{conv}_{\Theta_{\mathcal{X}}^7} \circ \mathfrak{r} \circ \mathrm{resconv}_{\mathfrak{r}}\left(\Theta_{\mathcal{X}}^4, \Theta_{\mathcal{X}}^5, \Theta_{\mathcal{X}}^6\right) \circ \mathfrak{r} \circ \mathrm{conv}_{\Theta_{\mathcal{X}}^3} \circ \mathfrak{r} \circ \mathrm{conv}_{\Theta_{\mathcal{X}}^2} \circ \mathfrak{r} \circ \mathrm{conv}_{\Theta_{\mathcal{X}}^1}\right](\mathrm{cat}(F,G)) \tag{32}$$

[9] One can also view the image as sliding over the weights, in which case the dilation controls how "quickly" the image slides, exactly analogous to the stride.

[10] `https://ezyang.github.io/convolution-visualizer/`

[11] This is thought to stabilize training by making it easy to learn identity mappings. Previous work had shown that very deep networks performed more poorly than shallower ones despite having greater capacity; by making the identity the "default" for each layer, residual learning ensures that a very deep network can't perform any worse than a shallower one.

ℓ (Layer)	k (Kernel size)	C_{out} (Channels)	σ (Stride)	d (Dilation)
1	(3, 3)	128	(1, 1)	**(2, 2)**
2	(3, 3)	128	**(2, 2)**	(1, 1)
3	(3, 3)	128	**(2, 2)**	(1, 1)
4	(3, 3)	128	(1, 1)	(1, 1)
5	(3, 3)	128	(1, 1)	(1, 1)
6	(3, 3)	128	(1, 1)	(1, 1)
7	**(1, 1)**	**8**	(1, 1)	(1, 1)

Table 1: The architecture parameters of the convolutional layers of $\mathcal{D}_\varphi$ and $\mathcal{G}_\theta$.

where $\mathcal{X}$ stands for either $\mathcal{D}$ or $\mathcal{G}$, and cat is the concatenation function that takes images $F \in D(n, c_F; s)$ and $G \in D(n, c_G; s)$ to the image $\text{cat}(F, G) \in D(n, c_F + c_G; s)$ defined by concatenating the tuple $G[i]$ to $F[i]$ to get $\text{cat}(F, G)[i]$ for all $i \in \mathbb{N}_0^s$.

Suppose the output $C_{\mathcal{X}}(F, G) \in D(2, 8; (H, W))$ for some H, W that depend on the shape s of F and G. If we "flatten" the output of $C_{\mathcal{X}}(F, G)$ into a vector $c_{\mathcal{X}}(F, G) \in \mathbb{R}^{8 \cdot W \cdot H}$, then we can apply a fully-connected neural network to this vector to get a final output with the right number of dimensions. For the discriminator Yan, *et al.* use the network

$$\text{H}_{\mathcal{D}}(x) = \mathfrak{s}\left(W_{\mathcal{D}}^2 \mathfrak{r}\left(W_{\mathcal{D}}^1 x + b_{\mathcal{D}}^1\right) + b_{\mathcal{D}}^2\right) \tag{33}$$

and for the generator they use

$$\text{H}_{\mathcal{G}}(x) = W_{\mathcal{G}}^2 \mathfrak{r}\left(W_{\mathcal{G}}^1 x + b_{\mathcal{G}}^1\right) + b_{\mathcal{G}}^2 \tag{34}$$

where $x \in \mathbb{R}^{8 \cdot W \cdot H}$, the first layer in both networks has 256 hidden units, the second and final layer in $\text{H}_{\mathcal{D}}$ has one unit (recall that a WGAN discriminator should be a real function of its input), and the second and final layer of $\text{H}_{\mathcal{G}}$ has t units (recall that our transformations are going to be parameterized by $\mathbb{R}^t$).

Bringing this all together, we can define

$$\mathcal{D}_\varphi(F, G) = \text{H}_{\mathcal{D}}\big(\text{flat } C_{\mathcal{D}}(F, G)\big) \qquad \varphi = \left(\Theta_{\mathcal{D}}, W_{\mathcal{D}}^1, W_{\mathcal{D}}^2, b_{\mathcal{D}}^1, b_{\mathcal{D}}^2\right) \tag{35}$$

$$\mathcal{G}_\theta(F, G) = \Big(F, G \circ \mathcal{T}\big(\text{H}_{\mathcal{G}}\big(\text{flat } C_{\mathcal{G}}(F, G)\big)\big)\Big) \qquad \theta = \left(\Theta_{\mathcal{G}}, W_{\mathcal{G}}^1, W_{\mathcal{G}}^2, b_{\mathcal{G}}^1, b_{\mathcal{G}}^2\right) \tag{36}$$

where flat refers to the flattening operation described above, and $\mathcal{T}(\lambda)$ is the transformation corresponding to the transformation parameter $\lambda \in \mathbb{R}^t$.

3.2 Spatial transformers

We need to propagate gradients through the composition of G with $\mathcal{T}(\lambda)$ in order to use gradient descent to optimize θ. In machine learning this operation is known as a *spatial transformer* layer [6]. As mentioned earlier, we can define this operation by using interpolation on G to interpret it as a continuous image, composing that continuous image with $\mathcal{T}(\lambda)$, then sampling the resulting continuous image to make it discrete again.

Definition 15. *Let $\mathcal{T}(\lambda)$ be a transformation of $[0,1]^n$ parameterized by $\lambda \in \mathbb{R}^t$, and let $r \in \mathbb{N}^n$. The **spatial transformer** with **grid size** r adapted to $\mathcal{T}(\,\cdot\,)$ is the function $\mathcal{S}_r(\lambda, \cdot\,)$ defined by*

$$\mathcal{S}_r(\lambda, G)[j] = \big(G_{\mathcal{K}}(\,\cdot\,) \circ \mathcal{T}(\lambda)\big)[j] = \sum_{i \in \mathbb{N}_0^s} G[i]\mathcal{K}\big(i, \mathcal{T}(\lambda)\big(\mathscr{G}_r(j)\big)\big) \qquad j \in \mathbb{N}_0^r, G \in D\,(n, c; s) \tag{37}$$

where $\mathcal{K}$ is an interpolation kernel used to define $G_{\mathcal{K}}(\,\cdot\,)$ (recall Definition 3). Note that $\mathcal{S}_r(\lambda, G) \in D\,(n, c; r)$ if $G \in D\,(n, c; s)$

In this definition I have emphasized the dependence of $\mathcal{S}$ on λ because we would like to be able to compute the Jacobian of $\mathcal{S}$ with respect to λ to use gradient descent in a neural network that involves a spatial transformer. Fix x; then $\lambda \mapsto \mathcal{T}(\lambda)(x)$ is a function mapping $\mathbb{R}^t \to \mathbb{R}^n$. Let $L(\lambda, x)$ be the Jacobian of this function. Then we can write the Jacobian of $\mathcal{S}$ with respect to λ as

$$\begin{aligned} J_{jk}(\lambda, G) = \frac{\partial \mathcal{S}[j]}{\partial \lambda_k} &= \sum_{i \in \mathbb{N}_0^s} G[i] \frac{\partial \mathcal{K}\left(i, \mathcal{T}(\lambda)\big(\mathscr{G}(j)\big)\right)}{\partial \lambda_k} \\ &= \sum_{i \in \mathbb{N}_0^s} G[i] \nabla \mathcal{K}\left(i, \mathcal{T}(\lambda)\big(\mathscr{G}(j)\big)\right) L_{\,\cdot\, k}(\lambda, x) \end{aligned} \tag{38}$$

where $L_{\,\cdot\, k}$ is the k-th column of L. For example, suppose that $\lambda = (A, b)$, where $A \in \mathbb{R}^{n \times n}$ and $b \in \mathbb{R}^n$, and let $\mathcal{T}(\lambda)(x) = Ax + b$. First, let's compute $L_{\,\cdot\, k}$, where $k = (\ell, m)$ points to $A_{\ell m}$. We have

$$L_{ik} = \frac{\partial (A_{i\,\cdot}\, x + b_i)}{\partial A_{\ell m}} = x_m \delta_{i\ell}$$

where δ is the Kronecker delta function, and $A_{i\,\cdot}$ is the i-th row of A. So $L_{\,\cdot\, k}$ is a vector with n components, all zero except for component ℓ equal to x_m. Thus, we get the Jacobian with respect to A of a spatial transformer $\mathcal{S}$ adapted to $\mathcal{T}$ via

$$\frac{\partial \mathcal{S}[j]}{\partial A_{\ell m}} = \sum_{i \in \mathbb{N}_0^s} G[i] \frac{\partial \mathcal{K}}{\partial x_\ell}\left(i, A\mathscr{G}(j) + b\right) \mathscr{G}(j)_m$$

Second, let's compute $L_{\,\cdot\, k}$ when k points to b_k. Then

$$L_{ik} = \frac{\partial (A_{i\,\cdot}\, x + b_i)}{\partial b_k} = \delta_{ik}$$

Thus, we get the Jacobian

$$\frac{\partial \mathcal{S}[j]}{\partial b_k} = \sum_{i \in \mathbb{N}_0^s} G[i] \frac{\partial \mathcal{K}}{\partial x_k}(i, A\mathscr{G}(j) + b)$$

Although this is not a common practical choice of interpolation kernel in machine learning, the following choice of $\mathcal{K}$ is nice for the purposes of this example. Let's use the Gaussian interpolation kernel

$$\mathcal{K}(i, x) = C \cdot \exp\left(-\frac{|x - \mathscr{G}(i)|^2}{2\sigma^2}\right)$$

for some suitable choice of constants C and σ. Then our Jacobian becomes

$$\frac{\partial \mathcal{S}[j]}{\partial A_{\ell m}} = \frac{1}{\sigma^2} \sum_{i \in \mathbb{N}_0^s} G[i] \mathcal{K}(i, A\mathscr{G}(j) + b) \big(\mathscr{G}(i)_\ell - \mathscr{G}(j)_\ell\big) \mathscr{G}(j)_m$$

$$\frac{\partial \mathcal{S}[j]}{\partial b_k} = \frac{1}{\sigma^2} \sum_{i \in \mathbb{N}_0^s} G[i] \mathcal{K}(i, A\mathscr{G}(j) + b) \big(\mathscr{G}(i)_k - \mathscr{G}(j)_k\big)$$

At this point, it should be fairly clear how this gradient computation can be carried out numerically (albeit somewhat inefficiently, perhaps). Usually the interpolation kernel is chosen to be either the nearest-neighbor kernel or the bilinear kernel. These kernels have a small support, which limits the number of terms in the summation to a very small number, thereby increasing the efficiency of this operation. There is also the successful Region of Interest Align kernel used in localization and segmentation tasks. It is a somewhat more sophisticated version of the bilinear kernel.

4 Experiments

Rather than write this section twice, I have written up the experimental portion of this project in Google Colab, an online service that makes it easy to edit, run, comment and, most importantly, share code[12]. You can find the project at this link[13], which should give you access to view, run, and edit the code and commentary of the project.

[12] This isn't a Google advertisement – Colab has its drawbacks, too.

[13] In case clicking on link didn't work: `https://colab.research.google.com/drive/1_rf-bgT5ZCrSlfNW0wNB_i82z4Np_rEE?usp=sharing`

References

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein GAN, 2017.

[2] G.E. Christensen and H.J. Johnson. Consistent image registration. *IEEE Transactions on Medical Imaging*, 20(7):568–582, 2001.

[3] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[4] A. Ardeshir Goshtasby. *2-D and 3-D Image Registration: For Medical, Remote Sensing, and Industrial Applications*. Wiley-Interscience, USA, 2005.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[6] Max Jaderberg, Karen Simonyan, Andrew Zisserman, and Koray Kavukcuoglu. Spatial transformer networks. *CoRR*, abs/1506.02025, 2015.

[7] Pingkun Yan, Sheng Xu, Ardeshir R. Rastinehad, and Bradford J. Wood. Adversarial image registration with application for MR and TRUS image fusion. *CoRR*, abs/1804.11024, 2018.