GANs for Data Augmentation

Eric Gentry and Jiahua You

Link to proposal
Link to presentation

Introduction

Neural networks can be effective, efficient prediction models, but they generally require a relatively large amount of training before they sufficiently converge.

The simplest approach to address this is to use multiple *epochs* through a training set, allowing the optimizer to see each training example multiple times. Unfortunately this can lead to overfitting: the optimizer might think it's found a correlation between input data and output label that holds across multiple training examples, but in fact it has just seen the noise pattern from a single example multiple times. Fortunately, this suggests a possible solution: at each epoch change the training examples so that their noise is less correlated between epochs.

These approaches to increase the effective size of your training data are termed "data augmentation" transformations (Krizhevsky et al. 2012). Common examples include affine/geometric transformations, and noise "whitening" (e.g. Krizhevsky 2009), but care must be taken to ensure that these transformations do not change the true label of the image. For example, when classifying handwritten characters, applying a horizontal reflection to the letter "p" would require changing the label to "q"---the label isn't invariant to horizontal reflections. On the other hand, astronomical images tend to have arbitrary axes, so the label should remain invariant to a horizontal reflection. But the brightness of the main astronomical object relative to the noise might be physically meaningful---this means the label might not be invariant to whitening transformations, which would have been valid for handwritten characters.

So rather than rely on domain experts to define and implement the appropriate transformations, it would be interesting to see whether neural networks could learn to indefinitely generate new examples given a label, based off a finite training set size.

Using Generative Adversarial Networks (GANs) to generate these new examples is one method that has shown promise recently, but it's still unclear how generally applicable this approach is. Shrivastava et al (2016) had success when their GAN was conditioned on a synthetic image, and the generated need only to learn to apply a noise model to the image. Antoniou et al 2017 relax the need to condition on an entire image, and instead only condition their GAN on a lower dimensional representation extracted from an image. Both sets of researchers find that training their target neural network on GAN-generated data can often improve results, but a key question remains:

1. Will this still work if we only condition on the target labels, rather than condition on synthetic images or low

dimensional representations of real images?

Data

We use images from the Hyper Suprime-Cam (HSC) survey (Aihara et al. 2017), and labels from a galaxy catalog of the COSMOS survey (Laigle et al. 2016). In particular we are interested in two labels for each galaxy:

1. distance from the Milky Way, and
2. mass of the galaxy.

From these, we create a derived label: is a galaxy closer than a distance threshold *and* within a certain low mass interval. Ultimately, we want a classifier that can predict this derived label for new galaxy images.

The training dataset we have is relatively small. It consists of roughly 2000 images, each of which has 3 channels, and is $(50px)^2$. These images are each centered on a specific galaxy, but there may be other background or foreground objects within the field-of-view.

These galaxies are not an unbiased sample of the galaxies within the available survey field---instead they have already been preselected as the galaxies most likely to be the targeted nearby and low-mass galaxies. Most of the details of this pre-selection are unimportant, but it is good to remember that it already took into bulk features like the total brightness and the average color of each galaxy. This makes our classification task more difficult, because the most useful discriminating information has already been used. This does not complicate our validation, as we validate using images with the same pre-selection function.

When training the GAN we will use all of the images, but when training the classifiers we will hold out 20% of the data as a validation set. We ensure that the same data is held out for the classifier trained on real images and the classifier trained on GAN-generated images.

Architecture

GAN

The main focus of our project is on the GAN. In particular, we build a conditional GAN that is conditioned on the two *continuous* labels: distance and mass (transformed as redshift and log stellar mass, both standardized to have mean=0 and standard deviation=1).

In this section we will broadly describe the GAN architecture and methods, but more detailed diagrams can be found in our presentation and the code can be found on github.

Generator

The generator[1] is a relatively standard conditional GAN. We concatenate the conditional labels with a noise vector (representing a latent set of features that

[1] https://github.com/egentry/galaxyCGAN/blob/9dbd59dad518a3454080a136d3ba4abdbc48b7fc/gan.py#L168

should be add diversity even if the conditional labels are fixed). We then feed this input through two fully connected layers, and then through two resize-convolution layers. After each layer *except the final layer* we apply batch normalization and RELU activation. We apply no activation to the final output layer.

While deconvolutional layers might be slightly more common, we choose to avoid them due to strong "checkerboard" artifacts in our GAN output. Following the advice of Odena et al. (2016), we instead used a resize layer (using bilinear interpolation) followed by *unstrided* convolutions.

Discriminator/Predictor

The discriminator component of our GAN was much less standard. In particular, it was both a "predictor" as well as a discriminator.[2]

The predictor component is uncommon but fairly straightforward. We feed in an image (real or generated), and it outputs a prediction for the distance and mass labels. It does this by first applying two convolutional layers (with batch normalization and LRELU activation after each), and then applying two fully connected layers (with batch normalization and LRELU activation between the layers, but not after the final layer).
The motivation for this predictor component comes from Ravanbakhsh et al. (2016), who were a mix of computer scientists and domain experts in galaxy imaging. They found that the traditional conditional GAN architecture (Mirza and Osindero 2014), which feeds the conditional label *into* the discriminator, performed well on categorical conditional labels but not *continuous* conditional labels. When Ravanbakhsh et al. (2016) switched to a predictor they claim they found better results. Anecdotally we observed the same thing, but we are unaware of any rigorous study documenting this or any theoretical justification for the predictor.

But we can't completely get rid of the discriminator. Ravanbakhsh et al. (2016) found that constructing a GAN with just a predictor and no discriminator often leads to mode collapse, which we confirmed in our early testing. So we followed their advice, and implemented *minibatch discrimination* (Salimans et al. 2016), which effectively looks for correlations *between* images in a given batch. As input features for the minibatch discrimination, we fed in the outputs from the final convolutional layer in the predictor *for an entire batch*. The minibatch discriminator then creates a new set of features for each image. The new features for each image are concatenated with the features for each image that were input into the minibatch discriminator. Finally, for each image, these combined features are passed through a single fully connected layer, with a sigmoid activation, representing the probability that a particular image is real or fake.

[2] https://github.com/egentry/galaxyCGAN/blob/9dbd59dad518a3454080a136d3ba4abdbc48b7fc/gan.py#L223

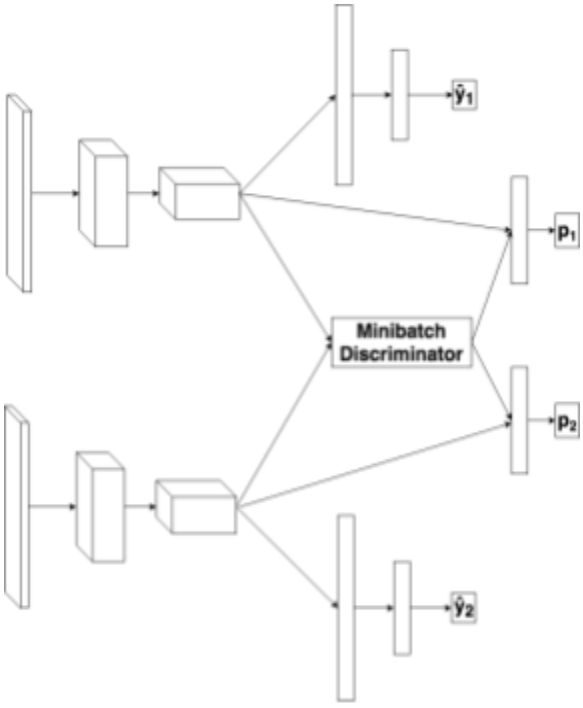

Figure 1. A diagram of the discriminator and predictor architecture, for an example batch size of 2. For each image, the network predicts the label ŷ, and assigns a discriminative probability p that the image is real. The prediction for ŷ is independent for each image in the batch; the discriminative probability of one image *does* depend on the other images in the batch.

Loss functions

We used two primary types of losses: an ℓ^2-based loss for ŷ and a entropy-based loss for p.[3] In particular for the generator we minimize:

$$\ell^2(y - \hat{y}(x_{fake}(y))) \; - \; \lambda \, log(p(x_{fake}(y))$$

where λ is a hyperparameter weighting the two flavors of loss.

Similarly, for the discriminator/predictor we minimize:

$$max[0, \; \ell^2(y - \hat{y}(x_{fake}(y))) \; - \; \ell^2(y - \hat{y}(x_{real}))]$$

$$- \; \lambda \, log(1 - p(x_{fake}(y))) \; - \; \lambda \, log(p(x_{real})) \, .$$

Ravanbakhsh et al. (2016) recommended[4] and briefly discuss this hinged-difference formulation, but don't clearly show why this is necessary.

Finally, we minimized this system using Adam (Kingma and Ba 2014). We used separate learning rates for the discriminator/predictor and the generator: .0001 and .0004 respectively. We used β_1=0.5 and β_2=0.999.

Classifier

Fortunately the classifier is pretty standard. We used 3 convolutional layers, with RELU activation, max pooling and dropout after each layer. Then we used 3 fully connected layers, with RELU activation between each and a sigmoid activation after the final layer.

The loss function for the classifier was the standard binary cross-entropy loss, where a label=1 corresponds to "*is* a nearby, low mass galaxy" and label=0 corresponds to "is *not* a nearby low mass galaxy".

We optimized the classifier using a learning rate of 0.001, β_1=0.9 and β_2=0.999.

[3] https://github.com/egentry/galaxyCGAN/blob/9dbd59dad518a3454080a136d3ba4abdbc48b7fc/gan.py#L323

[4] Actually they recommend a slightly different version in the text (where they change the `max` operator into a `min` and flip the 'real' and 'fake' terms), but we believe there to be an error in their text. The loss function they wrote doesn't match what the behavior seen in their figures. We used the form written in this report, which would better mesh with their figures, and which gave us much better results.

Results

GAN

Once trained (for 400-500 epochs) the GAN can make some galaxy images that look believable (see Figure 2), but it doesn't show clear evidence of capturing the correct trends with respect to the conditional labels (see Figure 3).

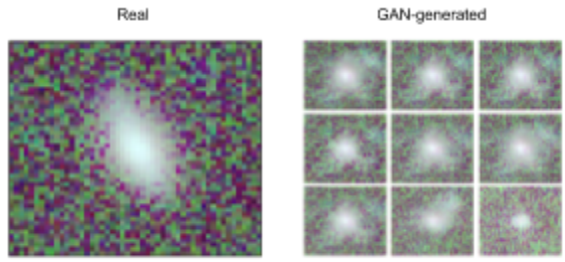

Figure 2. Comparison of real images and GAN-generated images.

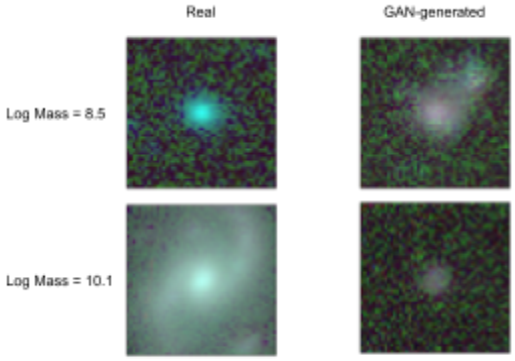

Figure 3. Comparison of how GAN-generated images scale with input conditional labels. Note: this image uses a slightly different stretch function from Figure 2, but that does not change the conclusions

The failure of our GAN to properly capture the correlation between images and the conditional labels suggests that our loss function is non-ideal. In particular, it might be that we need separate weighting terms for each label getting combined by the ℓ^2 loss. It could also be that we need a more complex loss function less susceptible to outliers (which we know exist in our data, leading to strong non-gaussianities in the label distributions). Finally, maybe it's simply a limitation of our small, pre-selected dataset.

Classifier

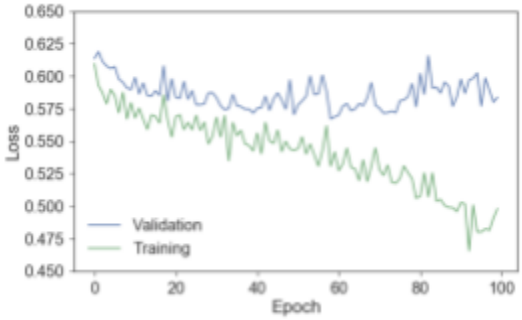

Figure 4. Learning curve of our classifier trained on real images and traditional data augmentation.

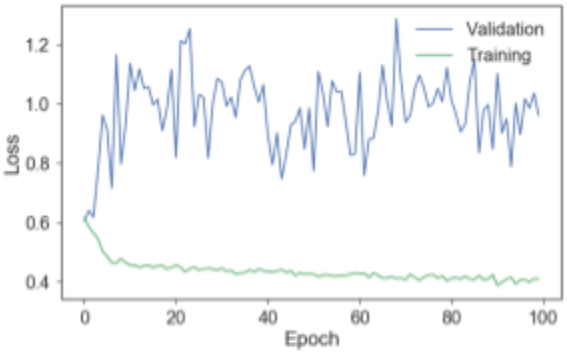

Figure 5. Learning curve for the classifier trained only on GAN-generated images. Noting the different y-axis scales, we see that the validation results are *mush* worse than seen in Figure 4, even though the *training* loss would suggest it is much better.

In Figure 4 we show the learning curve of our classifier trained on real data with traditional data augmentation. It exhibits the standard behavior: the validation loss initially decreases, but then starts to flatten out. Near the end of the training, the training loss continues to decline by the validation loss is starting to rise, indicating over-training. We're not surprised that there

can be overtraining, given the relatively small training set size.

The more surprising behavior is seen in Figure 5, the learning curve of the classifier trained purely on GAN-generated images. In that case, the validation loss basically never decreases; the network parameters are “optimized” to become *worse* than their random initializations. The training loss typically *thinks* that it’s doing better than in Figure 4, but that isn’t useful for us.

Ultimately, training on fake images and validating on real images leads to overtraining *almost immediately*.

Conclusions

Things we would try in the future:

- Training the GAN on a larger training set (even though it won’t be balanced the same as our validation set)
- More tuning to balance the different components of the loss function.

Answers to our main questions (from the introduction):

1. Training only on our GAN images led to *clearly* worse results than using traditional data augmentation.

Ultimately, using a conditional GAN for data augmentation *worsened* the performance of the classifier. This is not too surprising. Although Figure 2 shows that the GAN can sometimes make believable images, Figure 3 shows that it doesn’t reliably capture the expected dependence on the conditional labels. While we might be able to improve this performance with more work, the key result is that it led to *decreased* classification performance relative to traditional data augmentation methods.

For this reason, our recommendation is to avoid using GANs for data augmentation when you can’t condition the GAN on an input image or at least a lower dimensional encoding on an image. Trying to condition just on a continuous label did not provide enough information to allow the GAN to outperform using existing, non-generated images.