THE NEURAL PERSPECTIVE
DEEP LEARNING SIMPLIFIED



# IMPROVED TECHNIQUES FOR TRAINING GANS

Edit

In the first **GAN tutorial** we covered the fundamentals of training GANs. In this post we will continue of the same data but implement some improved techniques from this **paper**.

## MAIN ISSUE:

When training GANs our objective is to find the Nash equilibrium for two player mini-max game. The Nash equilibrium can be intuitively defined as both players wanting to continue their strategies regardless of what the other player is going. For GANs, the Nash equilibrium is when the cost for D is at a minimum with respect to \theta_D and the cost for G is at a minimum with respect to theta_G.

Traditionally we will use gradient descent but we should note that J_D = f(\theta_D, \theta_G) and J_G = f(\theta_D, \theta_G). Using gradient descent to lower J_D can increase J_G and vice verse. This doesn't help convergence.

## FEATURE MATCHING

The first improvement technique is feature matching. The objective of G is to minimize log(1-D(G(z)) which is same maximizing the output of D (log(D(G(z))). Instead of maximizing directly on the output of D, we should maximize on the activation outputs from an intermediate layer in D. Think of a CNN, the intermediate conv layers are the feature detectors whereas the final FC layers are just used for classification. So, likewise, if we train on D's intermediate layer outputs, we are essentially training G to really learn from the discriminative features instead of just the output.

So training will now involve minimizing the difference between D's intermediate layer and one of G's intermediate layers. This technique works very well empirically. The new objective looks like this:

$$||f(x) - f(G(z))|| = \sqrt{\sum_{i}^{N}(f(x_i) - f(G(z_i)))^2}$$

where f is some intermediate layer in D

The tensorflow implementation is very simple. Just return the activation outputs from one of the intermediate layers and use that to redesign the new objective for G. Complete code is under the repo in **feature_matching.py**

First, we need to return the activation outputs from an intermediate layer.

```
def mlp(inputs):

    """
    Pass the inputs through an MLP.
    D is an MLP that gives us P(inputs) is from training d
    G is an MLP that converts z to X'.
    """

    fc1 = tf.nn.tanh(linear(inputs, FLAGS.num_hidden_units
    fc2 = tf.nn.tanh(linear(fc1, FLAGS.num_hidden_units, s
    fc3 = tf.nn.tanh(linear(fc2, 1, scope='fc3'))
    return fc3, fc2
```

Then we need to change the objective of G to the new one for feature matching.

```
self.cost_G = tf.sqrt(tf.reduce_sum(tf.pow(self.fc2_D_X-sel
```

And also keep in mind that we now need to feed in batch_X for stepping for G as well. With the normal GAN (without feature matching) G only cares about D(G(z)) for it objective but now it needs to factor is D(G(z)) and D(X) since it's trying to reduce the difference between the intermediate layer activate outputs from both. So the new step function for G looks like this:

```
def step_G(self, sess, batch_z, batch_X):
    input_feed = {self.z: batch_z, self.X: batch_X}
    output_feed = [self.cost_G,
                   self.optimizer_G]

    outputs = sess.run(output_feed, input_feed)
```

- SEARCH -

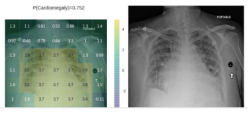
Search …

- FOLLOW ME ON TWITTER -

Tweets by @GokuMohandas

Goku Mohandas
@GokuMohandas

Your model may be performing really well by incorrectly focusing on confounding features (extraneous influencers in the data that aren't accounted for). Check out @johnrzech's post where x-ray stickers unintentionally influenced the classifications: medium.com/@jrzech/what-a…



Embed View on Twitter

- RECENT POSTS -

Update on Embeddings (Spring 2017)

Exploring Sparsity in Recurrent Neural Networks

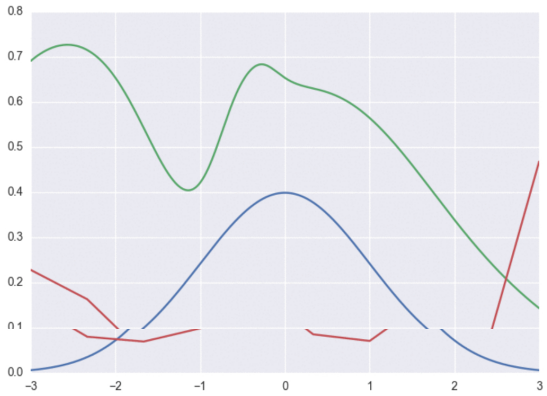
Question Answering from Unstructured Text by Retrieval and Comprehension

Overcoming Catastrophic Forgetting in Neural Networks

Opening the Black Box of Deep Neural Networks via Information

```
    return outputs[0], outputs[1]
```

The result it basically that a almost perfect decision boundary for D at 0.5. Compare this with the noisy decision boundary from the GAN implementation without feature matching. We are able to better learn the discriminative features for G but focusing on the intermediate layers rather than the binary output from D. Here is the transformation for our distributions with feature matching:




## MINIBATCH DISCRIMINATION

**Issue**: I'm going to be very verbose about it to clearly define the issue we are trying to solve because it can be a bit complicated. Let's think about learning a normal distribution. You have certain values of X that will produce high probability (pdf) in the normal distribution. First we pretrain D to match our p_data and now for the same values of X, D produces a high probability. Now it's time to train the GAN. We feed in random noise z into G which transforms into X'. If there X' are far away from the X that result in high P from D, then these X' will generate low P. If they are very similar to the X that result in high P from D, then these X' will generate a high P. G wants (D(G(z)) to be high and this happens when X' are similar to high P causing X. So as G is training, it will map more and more of it's random noise z to X' that very similar to high P resulting X. This is problematic because we are essentially causing G to produce X that converge to 1 max P producing point, which is certainly not learning the whole p_data distribution. This problem is called the **collapse of the generator.**

So why does this happen? It's because we are training D one point at a time. It receives X or X' and sees just that one point and has to determine probability P that the point is from the training set. When is sees the point is wants to see, it generate a high P. This is the crux of the issue leading to collapse of the gradient. **Note**: we still get the probability for each sample in the batch one at a time, but calculating that one probability now involves all the samples in the batch.

Solution is to factor in the entire batch. We will take our input, multiply by a tensor (trainable), get the absolute difference in L1 distance between this sample and all other samples in the batch for each row, apply a negative exponential operation and then take the resulting values for each row, sum them up and now we have our minibatch discrimination values. We will concat these to our normal outputs from the intermediate layer. **Note**: Dimension of the output now will change.

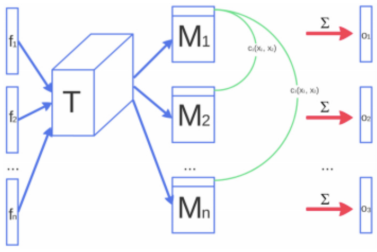



Figure 1: Figure sketches how minibatch discrimination works. Features $\mathbf{f}(\boldsymbol{x}_i)$ from sample $\boldsymbol{x}_i$ are multiplied through a tensor $T$, and cross-sample distance is computed.

*https://arxiv.org/pdf/1606.03498v1.pdf*

Why does this work? Since we are using all the other samples in the batch to influence D's prediction, we are effectively avoiding the collapse of the generator with this new side information.

**Results**: minibatch discrimination works really well and quickly to produce visually appealing results but empirically, feature matching results in better models (esp. for semi-supervised classification tasks).

## CONCLUSION

There are a few more techniques which were proven to be empirically successful in the paper but these two techniques are by far the ones I found to be most impactful. I may upload implementations for a few of them later (esp. virtual batch normalization). I will also be using many of these techniques in the DCGAN implementation.

## CODE:

**Github Repo** (Updating all repos, will be back up soon!)



*Posted in:* *Generative Adversarial Networks*, *Optimization/Architecture*
*Tagged:* *Tutorials*

← *Weights Initialization* *Improved Techniques for Training GANs* →

## 11 THOUGHTS ON "IMPROVED TECHNIQUES FOR TRAINING GANS"

Pingback: Training GANs: Better understanding and other improved techniques – Deep Learning IFT6266-H2017 UdeM Edit

**HOWARD** May 21, 2017 at 4:24 am EDIT REPLY →

The Github Repo is rot.How soon will it be accessible?

★ Liked by you

**PUAR HE** May 21, 2017 at 7:34 pm EDIT REPLY →

Hi Howard, I believe the author is working on pytorch repos for us as it will be clearer to use for a lot of these examples. until then I have still been able to follow the post and most of the code I need is within the post itself. I also have been using code tutorials online for GANs etc.

★ Liked by you

**GOKUMOHANDAS** May 22, 2017 at 8:33 pm EDIT REPLY →

Sorry about that, I discontinued the tensorflow repo for now because of external restrictions. I will be uploading PyTorch repos and also videos for a lot of these concepts soon (late summer).

★ Like

**ROD** June 1, 2017 at 8:32 pm EDIT REPLY →

Should it be without the sqrt root the feature matching equation?

L2 norm of x is ||x||_{2} = sqrt{sum(x_{i}^2)}

but is written ||x||_{2}^2 = sum(x_{i}^2)

★ Liked by you

**M** July 6, 2017 at 6:50 pm EDIT REPLY →

That is what I thought, too.

★ Like

Pingback: Multi View Image Generation | Exploring deep learning Edit

Pingback: Multi-View Image Generation | Exploring deep learning Edit

Pingback: An applied introduction to generative adversarial networks | | Crypto War - Bitcoin Guide Edit

Pingback: An applied introduction to generative adversarial networks – Cloud Data Architect Edit

Pingback: Uma introdução aplicada a redes adversárias generativas – iCrowdNewswire Portuguese Edit

Pingback: Generative Adversarial Networks – An Experiment with Training Improvement Techniques | high dimensional space Edit

## LEAVE A REPLY

Enter your comment here...