# Conditional Generative Adversarial Nets in TensorFlow

We have seen the Generative Adversarial Nets (GAN) model in the previous post (/techblog/2016/09/17/gan-tensorflow/). We have also seen the arch nemesis of GAN, the VAE and its conditional variation: Conditional VAE (CVAE). Hence, it is only proper for us to study conditional variation of GAN, called Conditional GAN or CGAN for short.

## CGAN: Formulation and Architecture

Recall, in GAN, we have two neural nets: the generator $G(z)$ and the discriminator $D(X)$. Now, as we want to condition those networks with some vector $y$, the easiest way to do it is to feed $y$ into both networks. Hence, our generator and discriminator are now $G(z, y)$ and $D(X, y)$ respectively.

We can see it with a probabilistic point of view. $G(z, y)$ is modeling the distribution of our data, given $z$ and $y$, that is, our data is generated with this scheme $X \sim G(X \mid z, y)$.

Likewise for the discriminator, now it tries to find discriminating label for $X$ and $X_G$, that are modeled with $d \sim D(d \mid X, y)$.

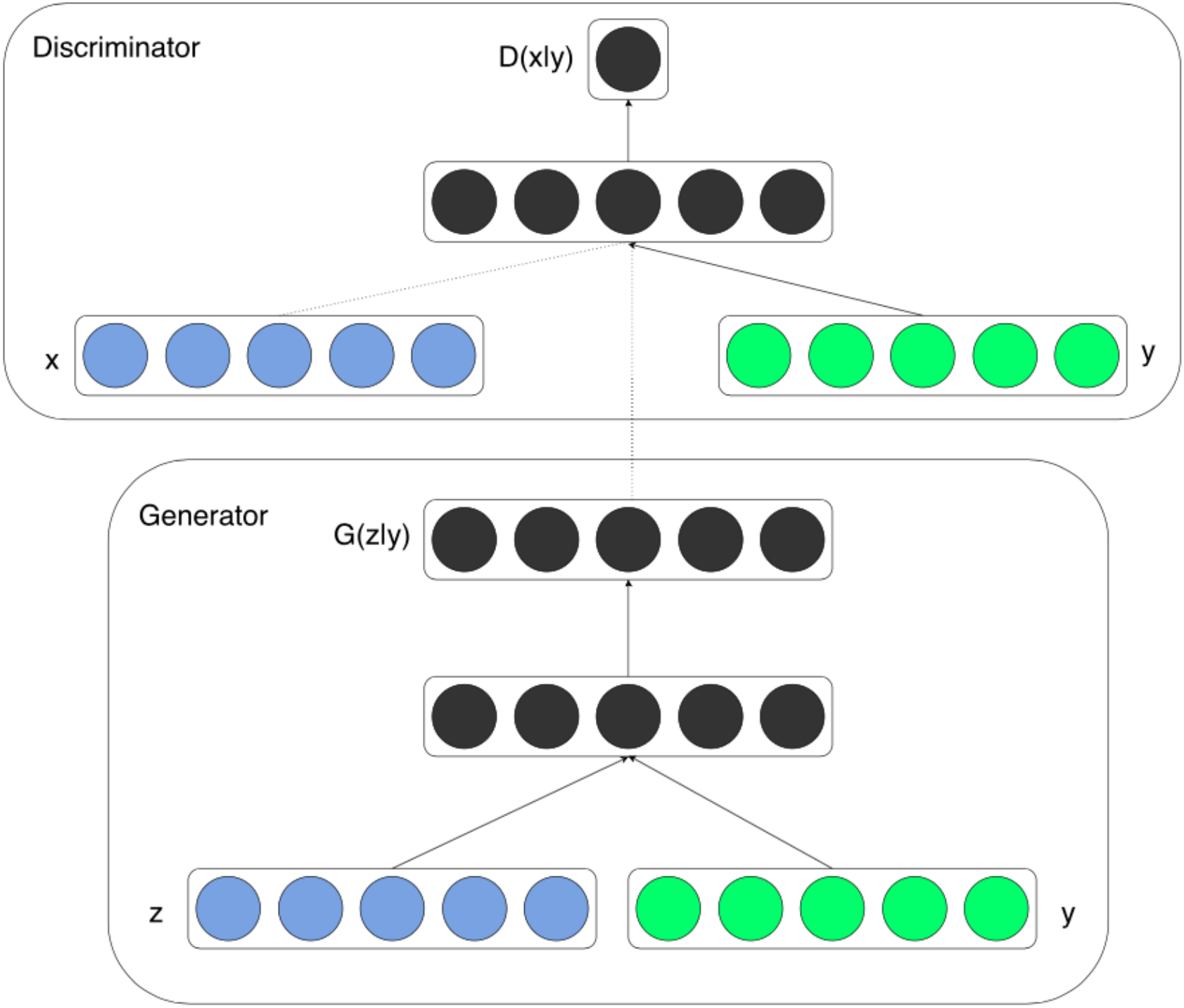
Hence, we could see that both $D$ and $G$ is jointly conditioned to two variables $z$ or $X$ and $y$.

Now, the objective function is given by:

$$\min_G \max_D V(D, G) = \mathop{\mathbb{E}}_{x \sim p_{data}(x)} [\log D(x, y)] + \mathop{\mathbb{E}}_{z \sim p_z(z)} [\log(1 - D(G(z, y), y))]$$

If we compare the above loss to GAN loss, the difference only lies in the additional parameter $y$ in both $D$ and $G$.

The architecture of CGAN is now as follows (taken from [1]):


Discriminator
D(xly)
x
y
Generator
G(zly)
z
y

In contrast with the architecture of GAN, we now has an additional input layer in both discriminator net and generator net.

## CGAN: Implementation in TensorFlow

I'd like to direct the reader to the previous post about GAN (/techblog/2016/09/17/gan-tensorflow/), particularly for the implementation in TensorFlow. Implementing CGAN is so simple that we just need to add a handful of lines to the original GAN implementation. So, here we will only look at those modifications.

The first additional code for CGAN is here:

```
y = tf.placeholder(tf.float32, shape=[None, y_dim])
```

We are adding new input to hold our variable we are conditioning our CGAN to.

Next, we add it to both our generator net and discriminator net:

```
def generator(z, y):
    # Concatenate z and y
    inputs = tf.concat(concat_dim=1, values=[z, y])

    G_h1 = tf.nn.relu(tf.matmul(inputs, G_W1) + G_b1)
    G_log_prob = tf.matmul(G_h1, G_W2) + G_b2
    G_prob = tf.nn.sigmoid(G_log_prob)

    return G_prob


def discriminator(x, y):
    # Concatenate x and y
    inputs = tf.concat(concat_dim=1, values=[x, y])

    D_h1 = tf.nn.relu(tf.matmul(inputs, D_W1) + D_b1)
    D_logit = tf.matmul(D_h1, D_W2) + D_b2
    D_prob = tf.nn.sigmoid(D_logit)

    return D_prob, D_logit
```

The problem we have here is how to incorporate the new variable $y$ into $D(X)$ and $G(z)$. As we are trying to model the joint conditional, the simplest way to do it is to just concatenate both variables. Hence, in $G(z, y)$, we are concatenating $z$ and $y$ before we feed it into the networks. The same procedure is applied to $D(X, y)$.

Of course, as our inputs for $D(X, y)$ and $G(z, y)$ is now different than the original GAN, we need to modify our weights:

```
# Modify input to hidden weights for discriminator
D_W1 = tf.Variable(shape=[X_dim + y_dim, h_dim]))

# Modify input to hidden weights for generator
G_W1 = tf.Variable(shape=[Z_dim + y_dim, h_dim]))
```

That is, we just adjust the dimensionality of our weights.

Next, we just use our new networks:

```
# Add additional parameter y into all networks
G_sample = generator(Z, y)
D_real, D_logit_real = discriminator(X, y)
D_fake, D_logit_fake = discriminator(G_sample, y)
```

And finally, when training, we also feed the value of $y$ into the networks:

```
X_mb, y_mb = mnist.train.next_batch(mb_size)

Z_sample = sample_Z(mb_size, Z_dim)
_, D_loss_curr = sess.run([D_solver, D_loss], feed_dict={X: X_mb, Z: Z_sample, y:y_mb})
_, G_loss_curr = sess.run([G_solver, G_loss], feed_dict={Z: Z_sample, y:y_mb})
```

As an example above, we are training our GAN with MNIST data, and the conditional variable $y$ is the labels.

## CGAN: Results

At test time, we want to generate new data samples with certain label. For example, we set the label to be 5, i.e. we want to generate digit “5”:

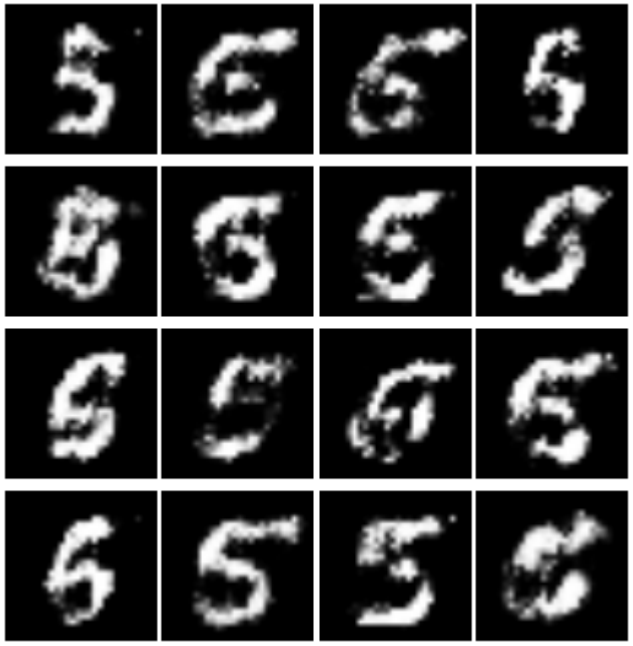
```
n_sample = 16
Z_sample = sample_Z(n_sample, Z_dim)

# Create conditional one-hot vector, with index 5 = 1
y_sample = np.zeros(shape=[n_sample, y_dim])
y_sample[:, 5] = 1

samples = sess.run(G_sample, feed_dict={Z: Z_sample, y:y_sample})
```

Above, we just sample $z$, and then construct the conditional variables. In our example case, the conditional variables is a collection of one-hot vectors with value 1 in the 5th index. The last thing we need to is to run the network with those variables as inputs.

Here is the results:



Looks pretty much like digit 5, right?

If we set our one-hot vectors to have value of 1 in the 7th index:

Those results confirmed that have successfully trained our CGAN.

## Conclusion

In this post, we looked at the analogue of CVAE for GAN: the Conditional GAN (CGAN). We show that to make GAN into CGAN, we just need a little modifications to our GAN implementation.

The conditional variables for CGAN, just like CVAE, could be anything. Hence it makes CGAN an interesting model to work with for data modeling.

The full code is available at my GitHub repo: https://github.com/wiseodd/generative-models (https://github.com/wiseodd/generative-models).

## References


1. Mirza, Mehdi, and Simon Osindero. “Conditional generative adversarial nets.” arXiv preprint arXiv:1411.1784 (2014).



← PREVIOUS POST (/TECHBLOG/2016/12/21/FORWARD-REVERSE-KL/)

NEXT POST → (/TECHBLOG/2017/01/01/MLE-VS-MAP/)

(/feed.xml) (https://github.com/wiseodd)