Implementing GANs

Duke MLSS

Alex Lew

Probability Distributions

$$x \sim p(x)$$

x "is drawn from" $p(x)$

x "is a sample from" $p(x)$

$p(x)$ a distribution assigning 50% probability to H, 50% to T

x a random variable, equal to H or T

Probability Distributions

$$x \sim p(x)$$

x "is drawn from" $p(x)$

x "is a sample from" $p(x)$

p(x) a distribution assigning some small probability to each possible setting of pixel values in a 24-bit-color 100x100 image.

x a random variable, equal to some 100x100 image.

Generative modeling

Approximate $p(x)$ that generated the data

Do we want to…

- evaluate relative likelihoods of different data points?
- compute conditional distributions?
- do inference on latent variables?
- produce new samples (perhaps conditioned on something)?

Note: we have no "labeled" $p(x)$ values, just lots of x

Generative modeling

For today:

Generate New Samples

(create “realistic” images, sounds, etc.)

Sampling

How do we generate a sample from a distribution?

Method 1: Go out and flip a coin.
(slow, expensive)

p(x) a distribution assigning 50% probability to H, 50% to T
x a random variable, equal to H or T

Sampling

How do we generate a sample from a distribution?

Method 2: *Simulate* a sample.

```
np.random.uniform(0,1)
```

Source of (pseudo-)randomness

```
def G(z):
  if z > 0.5:
    return "Heads"
  else:
    return "Tails"
```

Deterministic mapping

Sample from p(x)

Sampling

But sometimes, we don't know exactly what G(z) should be.

Sampling

But sometimes, we don't know exactly what G(z) should be.
We can estimate *parameters* from *data*.

```python
def G(z):
  if z > theta:
    return "Heads"
  else:
    return "Tails"
```

```python
np.random.uniform(0,1)
```

Source of (pseudo-)randomness

Deterministic mapping

Sample from p(x)

Sampling

How do we generate a sample from a distribution?

Method 1: Go out and take a picture.
(slow, expensive)

p(x) a distribution assigning some small probability to each possible setting of pixel values in a 24-bit-color 100x100 image.

x a random variable, equal to some 100x100 image.

Sampling

How do we generate a sample from a distribution?

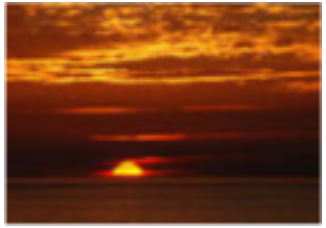
Method 2: *Simulate* a sample.

```
def G(z):
    # hmmmm…
    # help?
```

```
np.random.uniform(0,1)
```

Source of (pseudo-)randomness

Deterministic mapping



Sample from p(x)

How to write **G**?

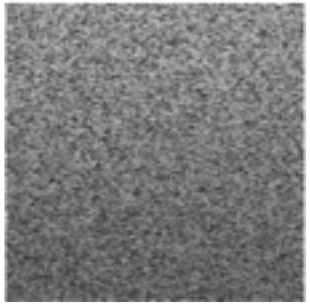
It's just a function: learn it!

```
def G(z):
  return MLP(z, theta)
```

```
np.random.uniform(0,1)
```

Source of (pseudo-)randomness

Deterministic mapping with randomly initialized parameters



Sample from p(x)

Learn **G**

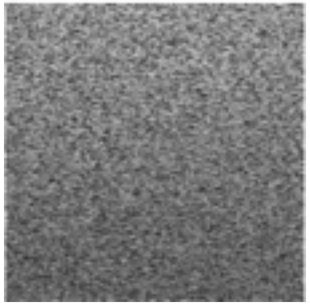
But with what loss function?

```
def G(z):
  return MLP(z, theta)
```

```
np.random.uniform(0,1)
```

Source of (pseudo-)randomness

Deterministic mapping with randomly initialized parameters



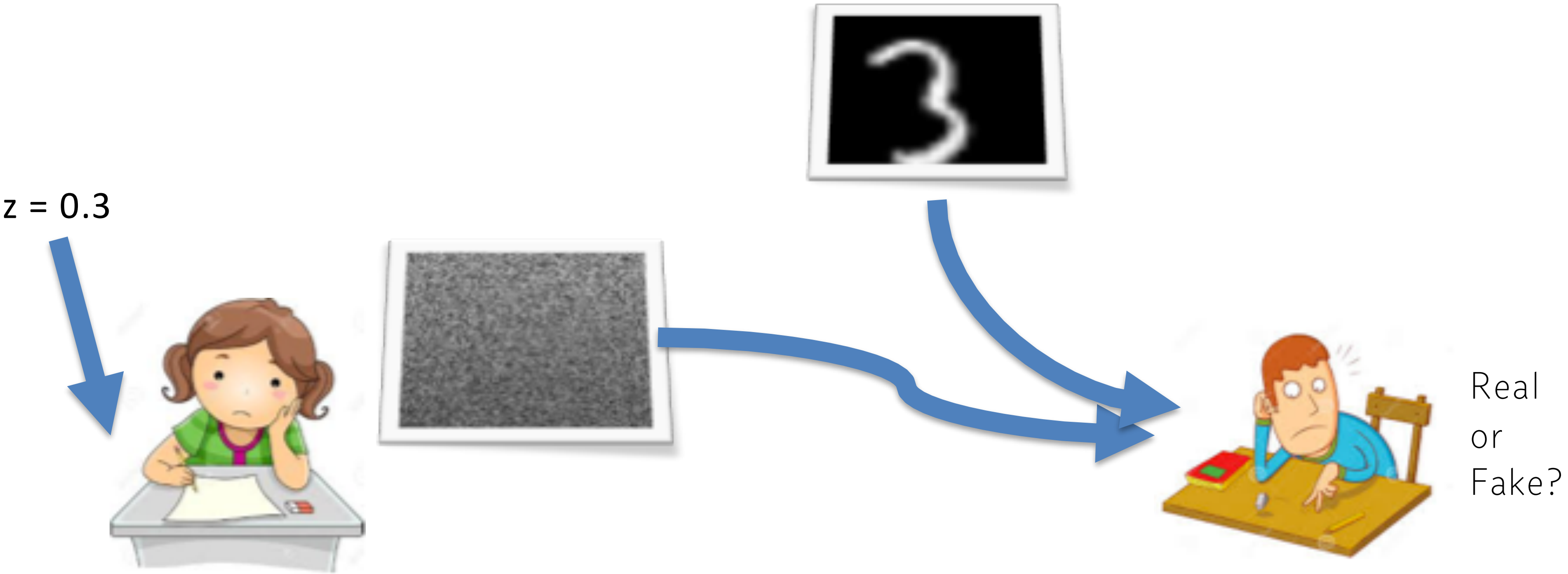
Sample from p(x)

Learn **G**

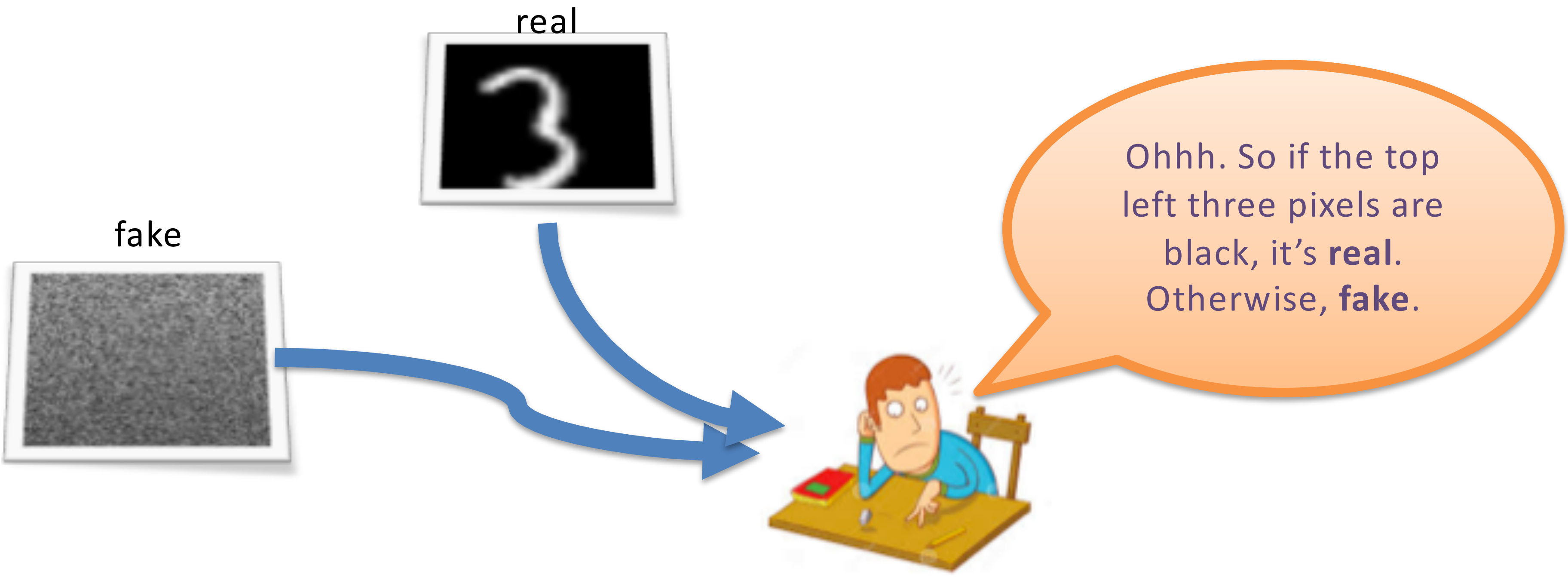
How do you teach **G** with no (in, out) examples?

I know my answer isn't right, but what should it have been for z=0.3?

Get two students (neural networks)



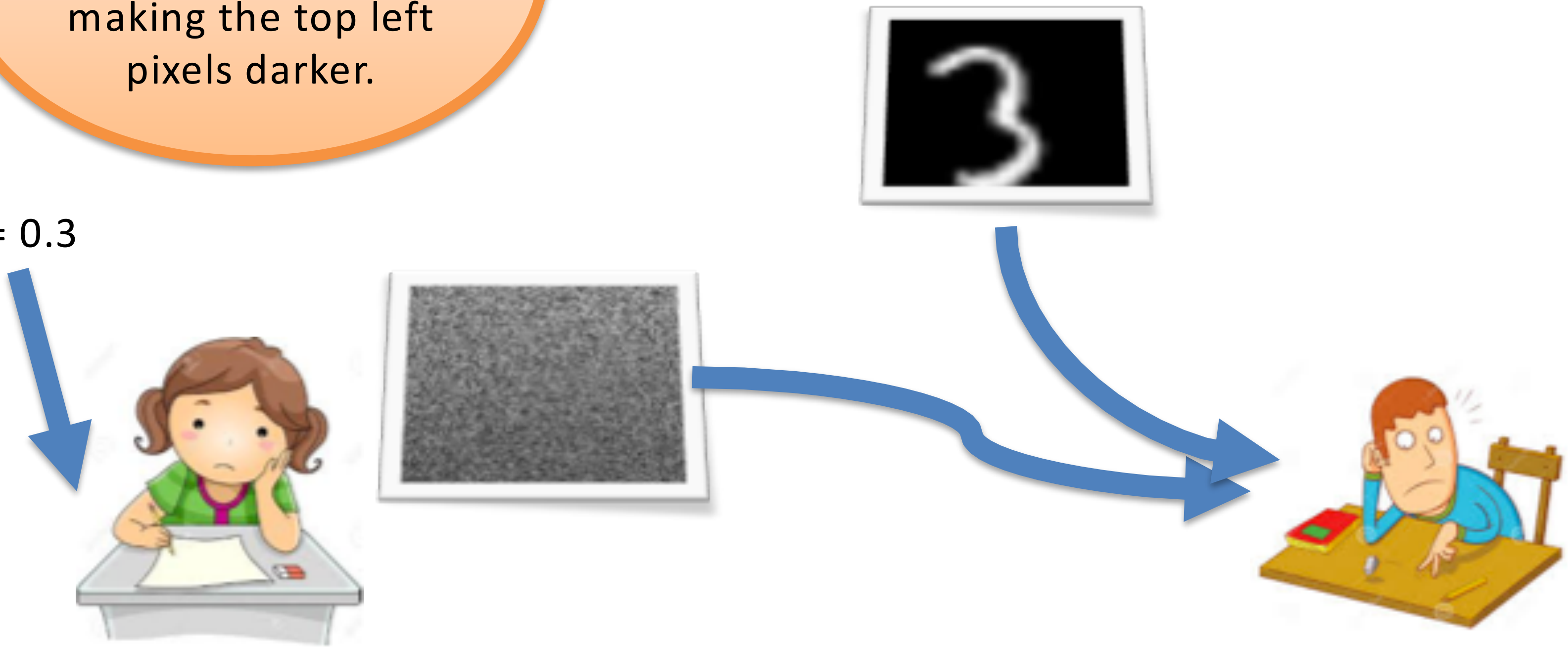
Get two students (neural networks)



Get two students (neural networks)

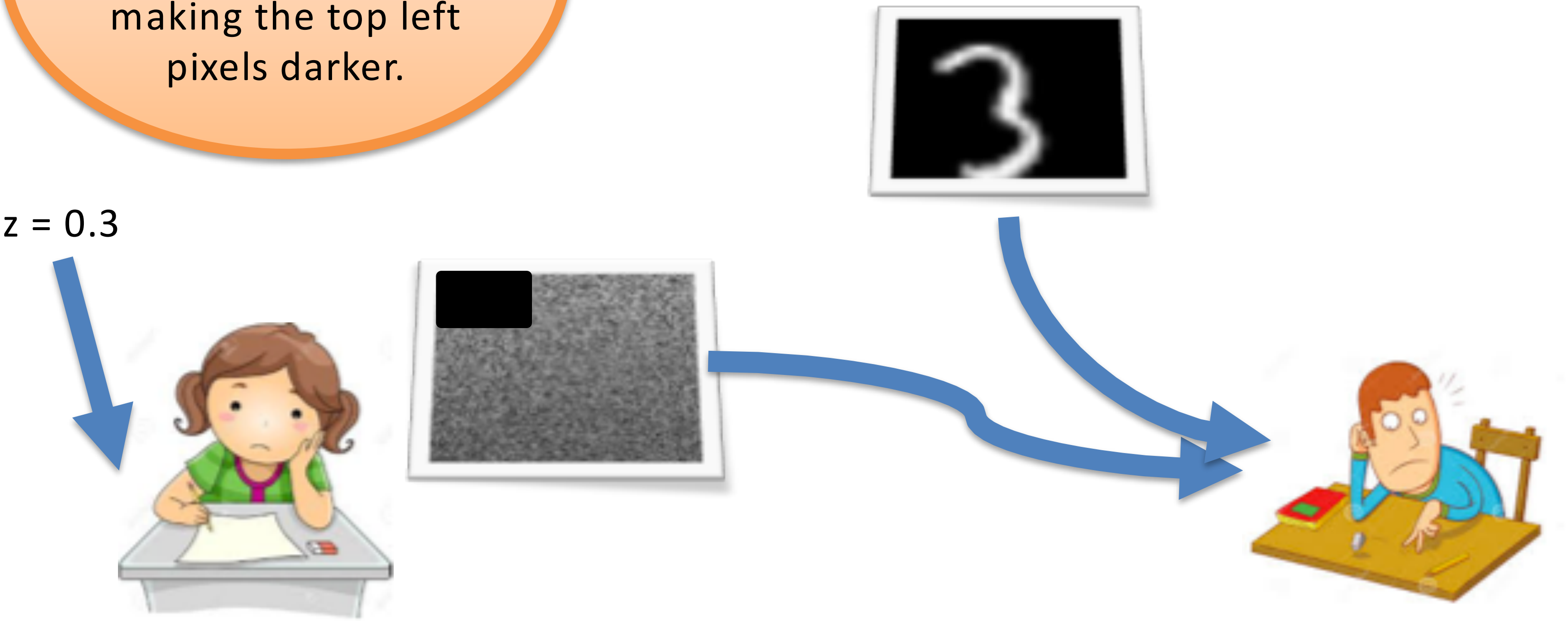
You're failing to
fool the
discriminator. Try
making the top left
pixels darker.

z = 0.3

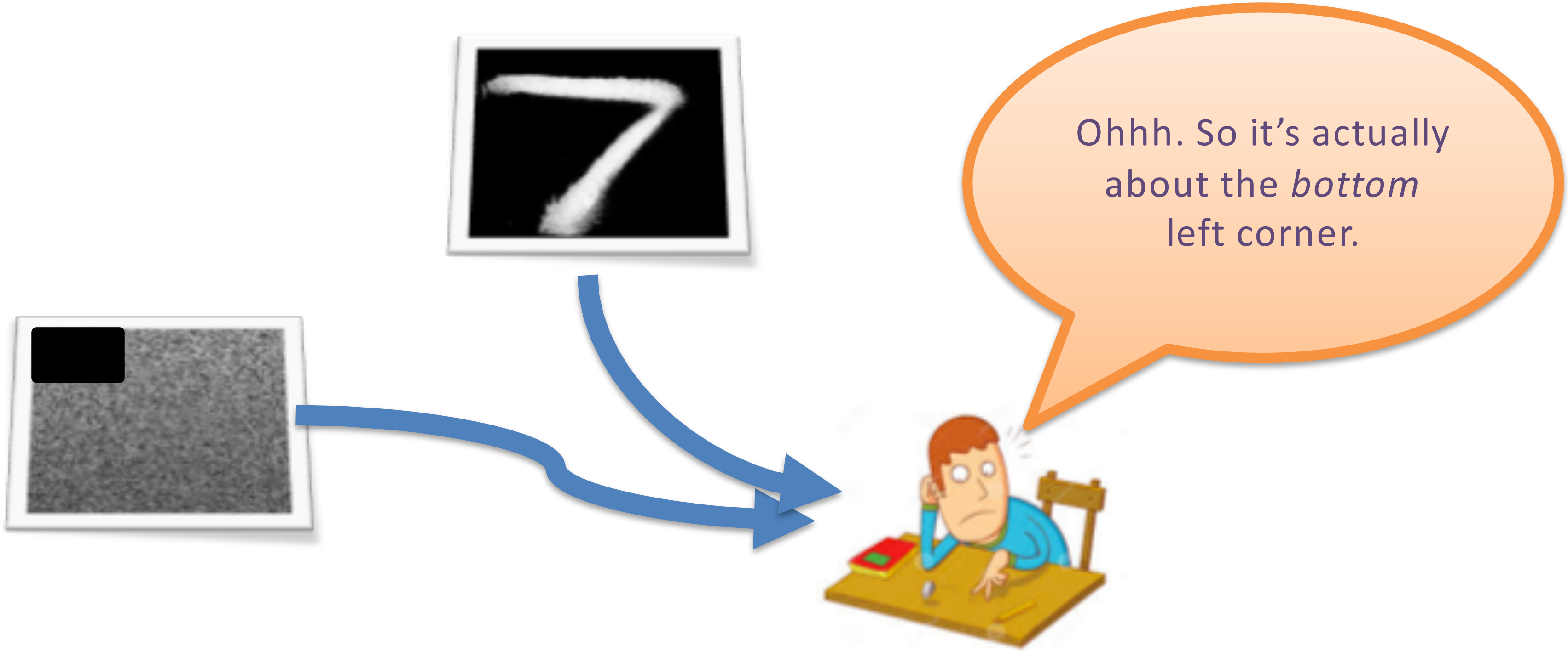


Get two students (neural networks)

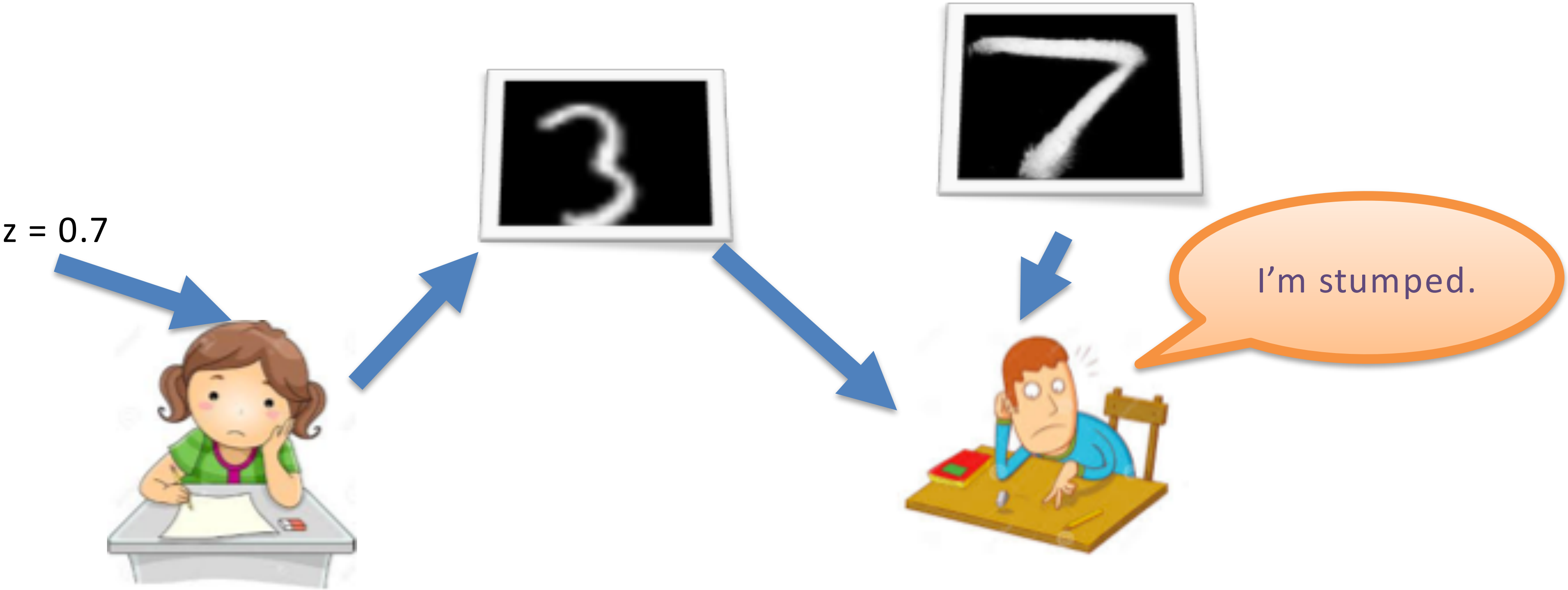
You're failing to fool the discriminator. Try making the top left pixels darker.



Get two students (neural networks)



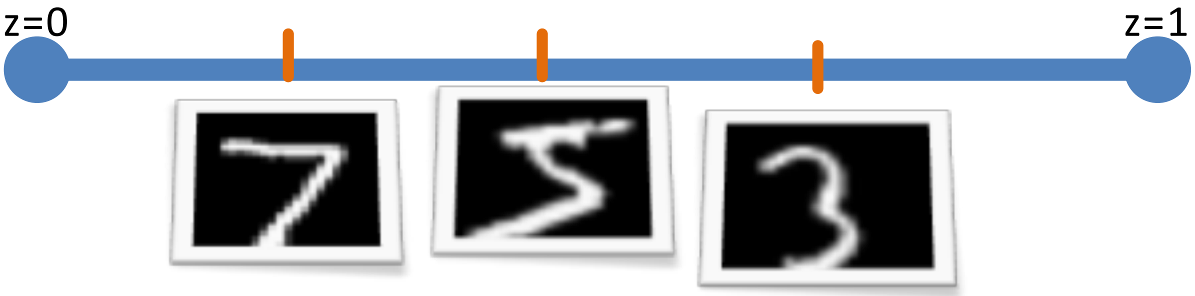
Many iterations later…



The importance of *z*

The importance of z

The importance of *z*

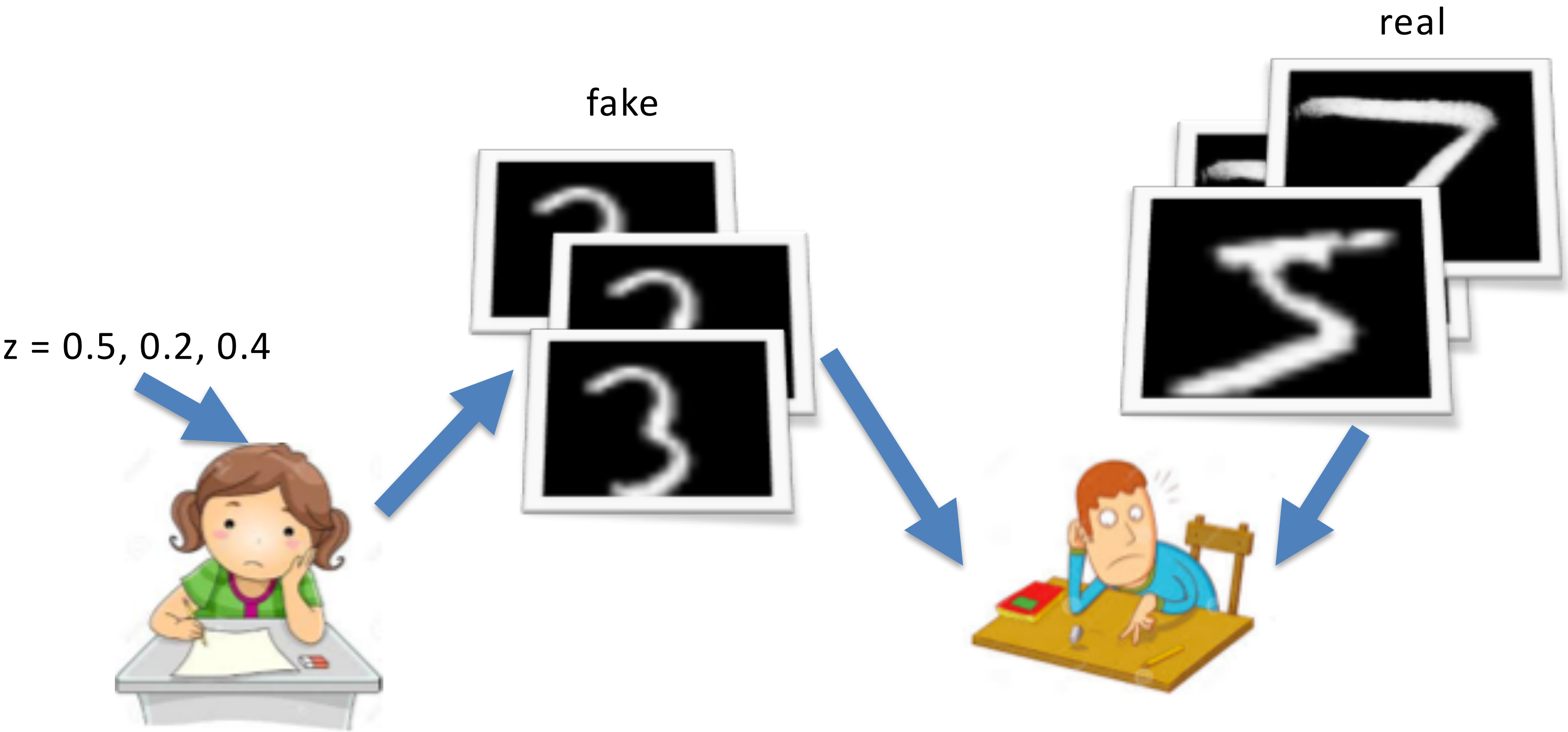


G a *deterministic function* from *z* to images

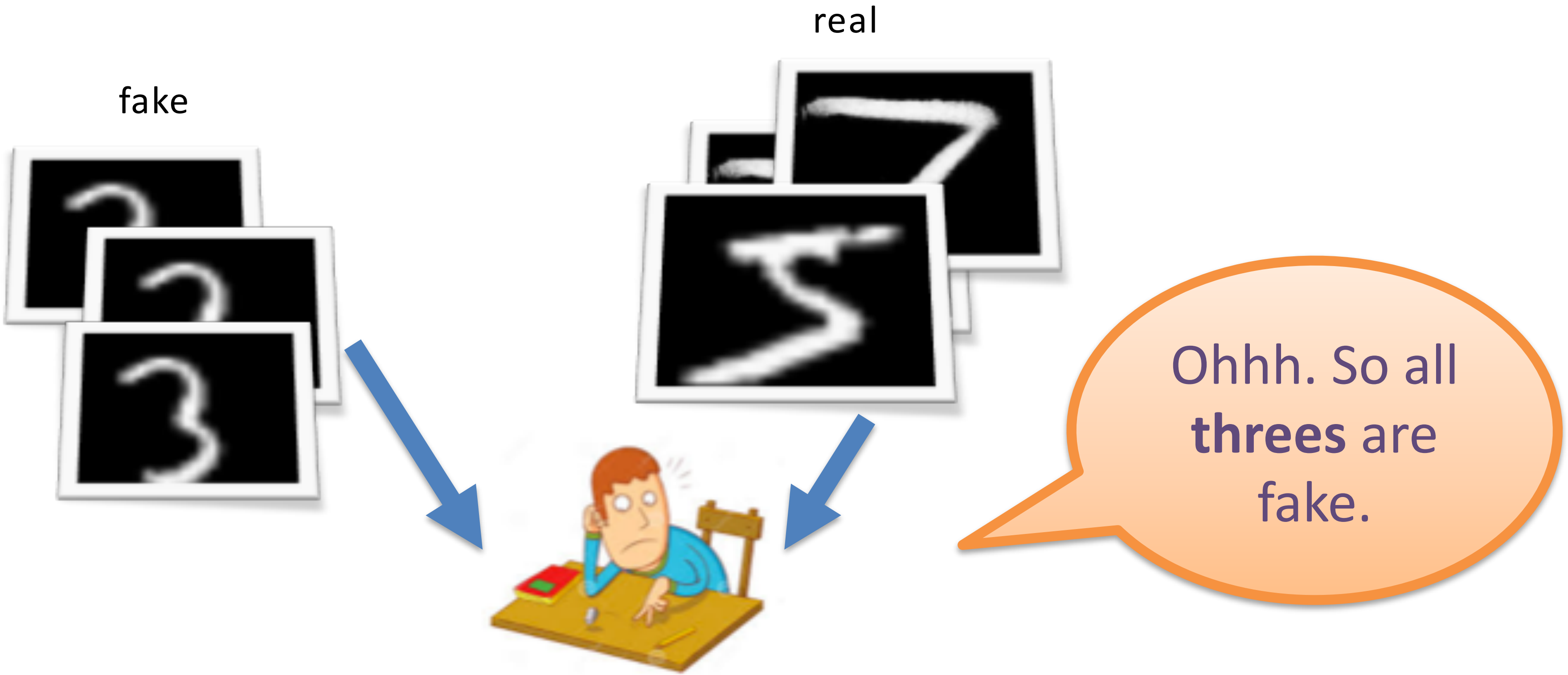
The importance of *z*

What if G learns to “play it safe” and always generate similar images?

What if G always writes the same thing?

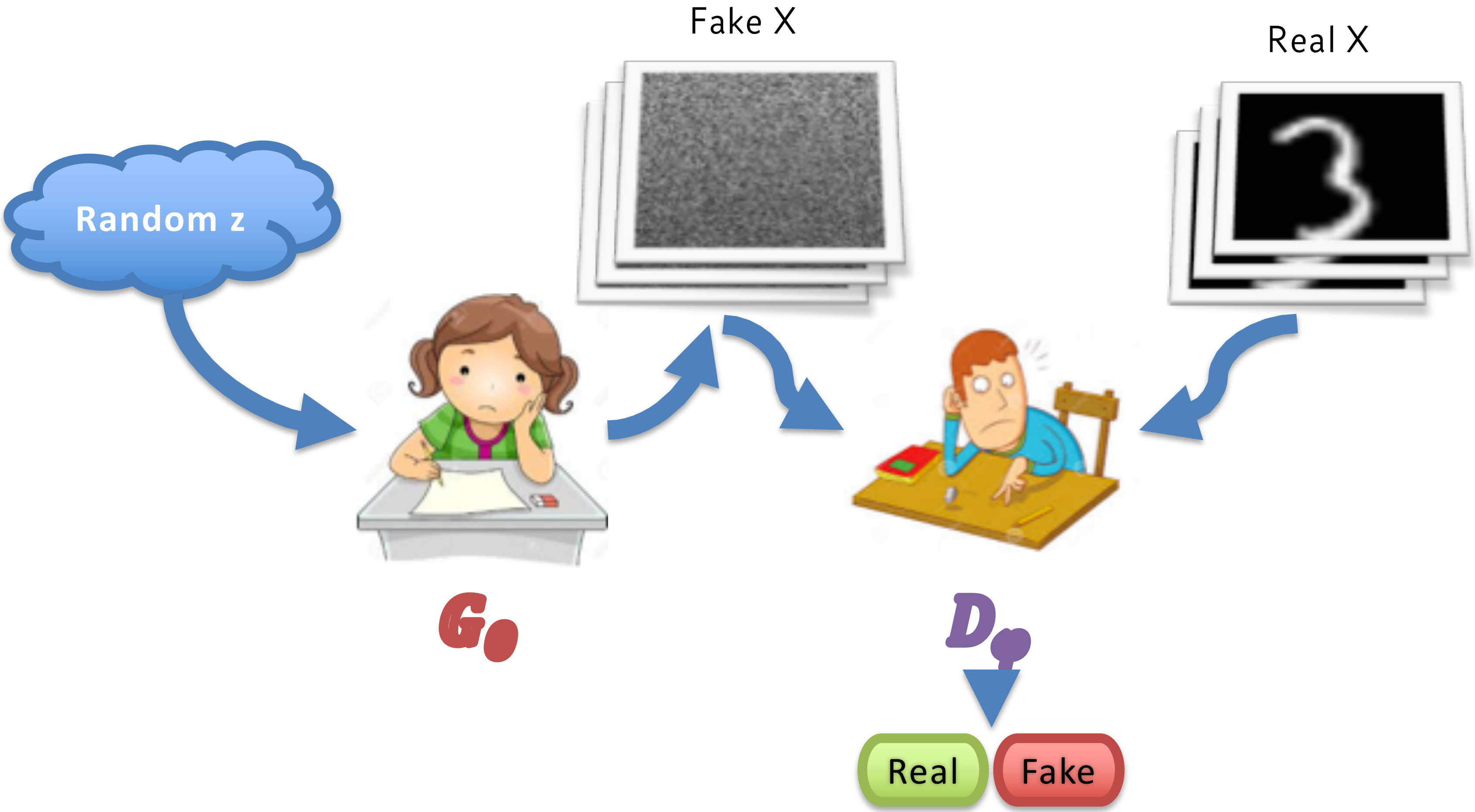


What if G always writes the same thing?



Main takeaway

We don’t know how to teach G to draw. But with two students, we can make them teach each other!



Fake X
Real X
Random z
G
D
Real
Fake

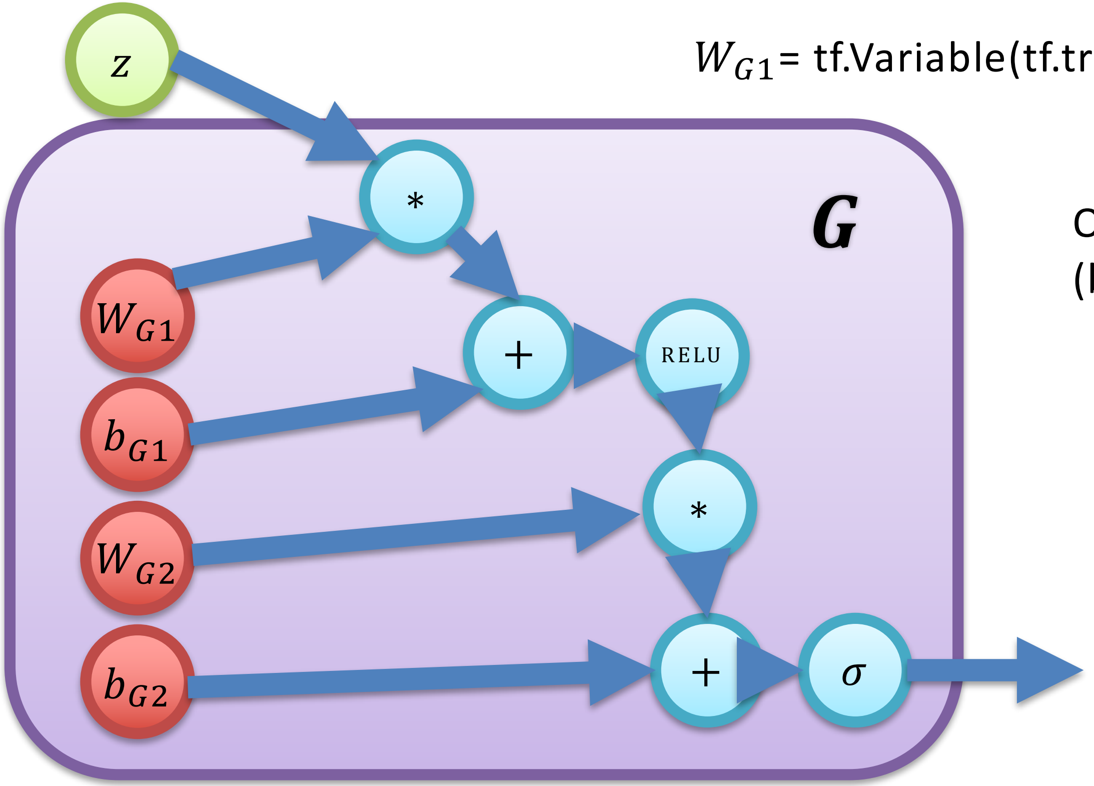
Implementation subtleties

z = tf.placeholder([50, 20])

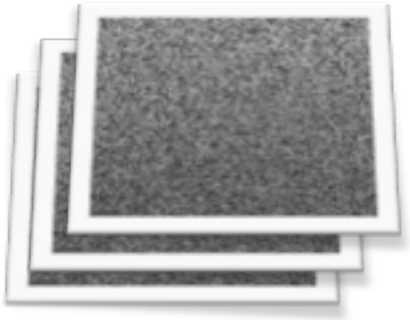
- We provide *multiple* random numbers (say, 20) to inform G in creating *each* image.
- We ask G to create a whole "batch" (say, 50) of fake images each batch.

Architecture of $\boldsymbol{G}$: 20 → 256 → 784

W_{G1}= tf.Variable(tf.truncated_normal([20, 256], stddev=0.1)



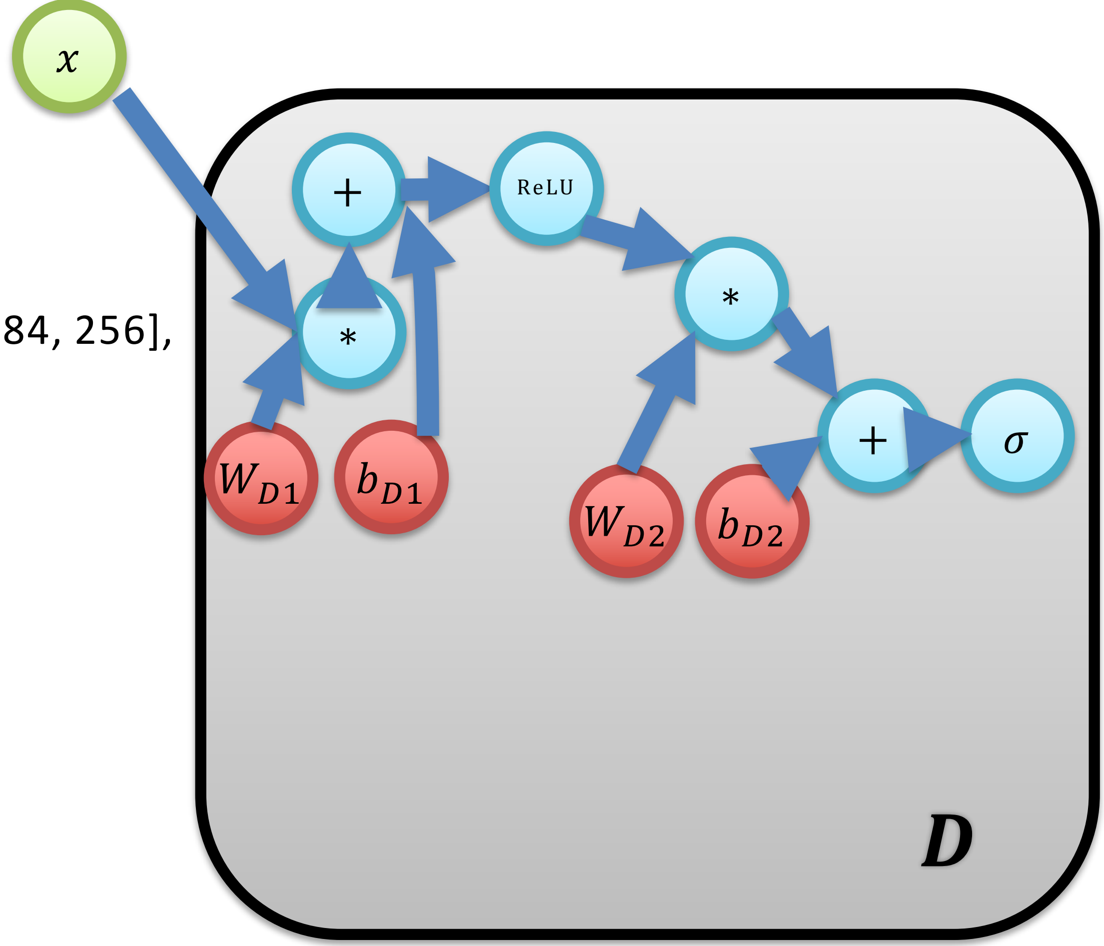
Our output: values between 0 and 1 (black and white) for each pixel.

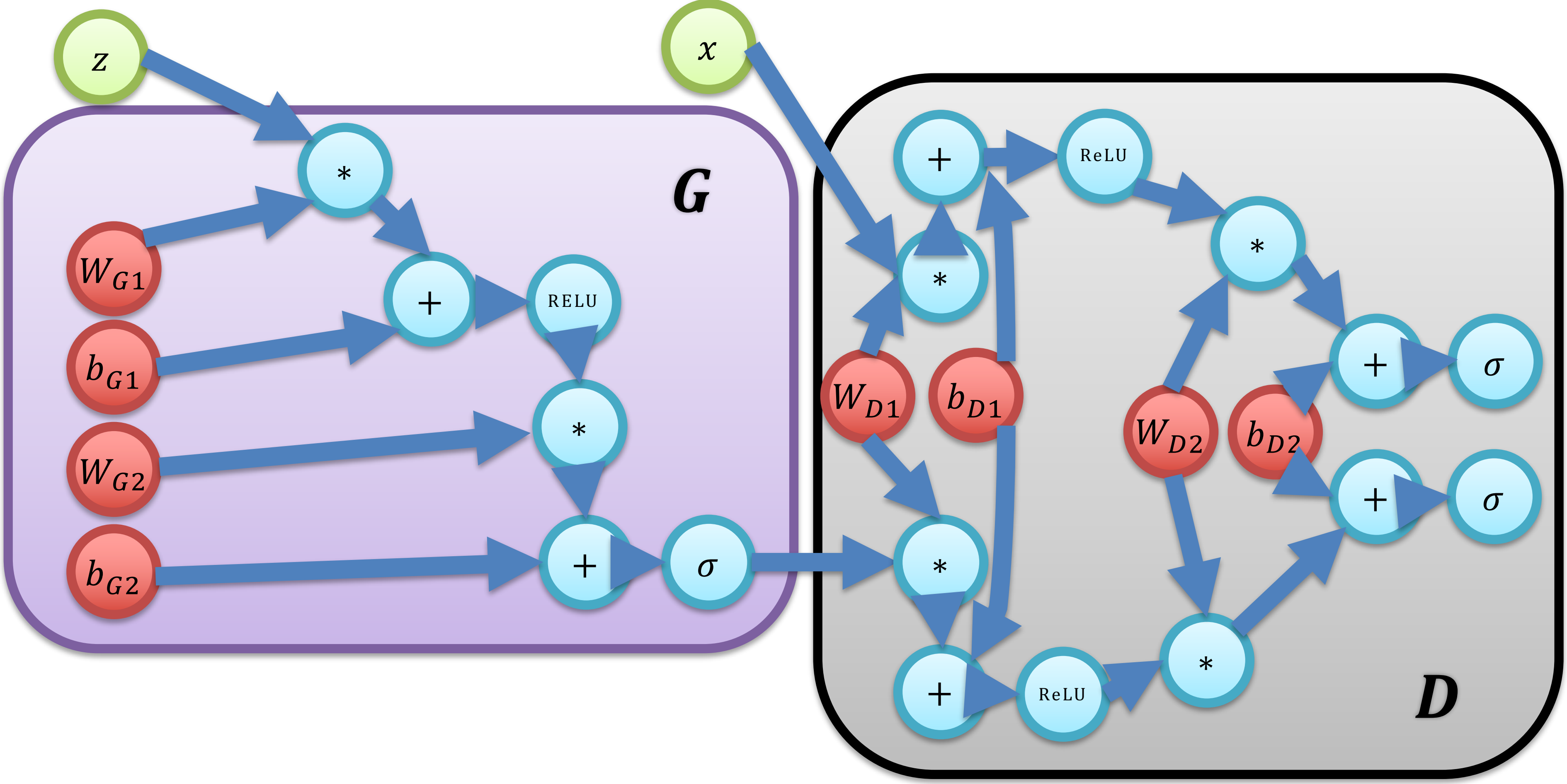


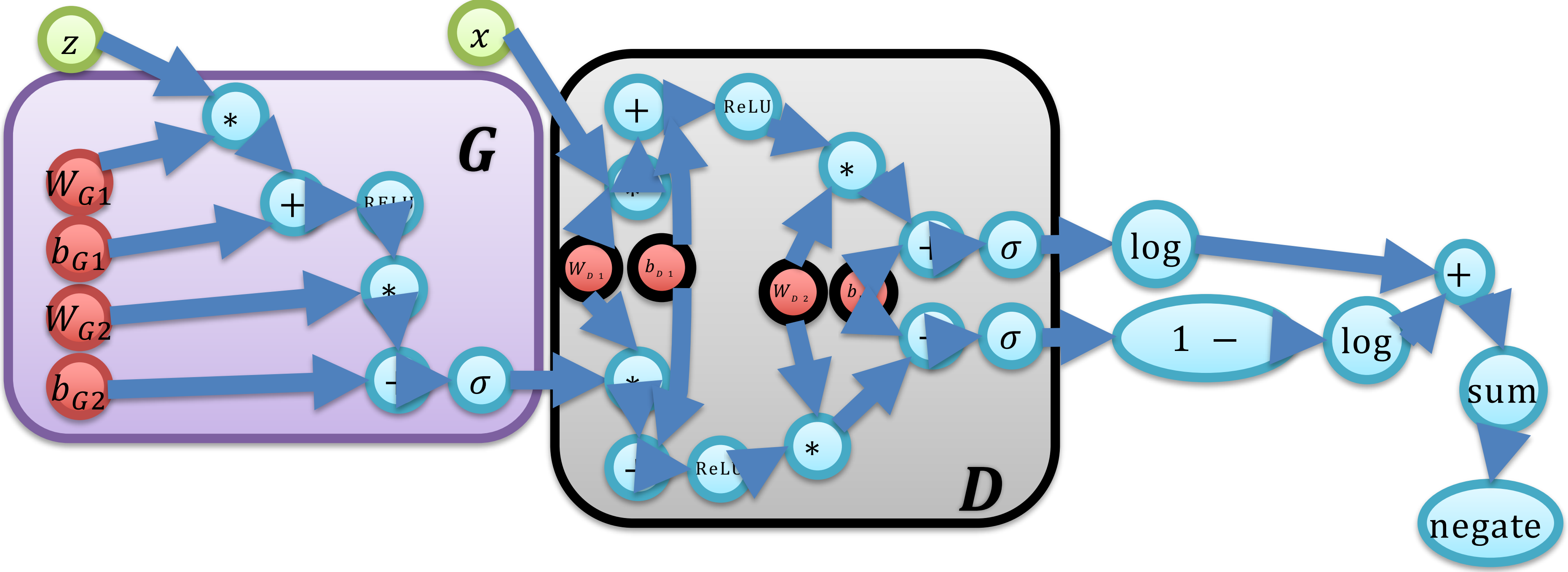
Architecture of **D**:

784 → 256 → 1

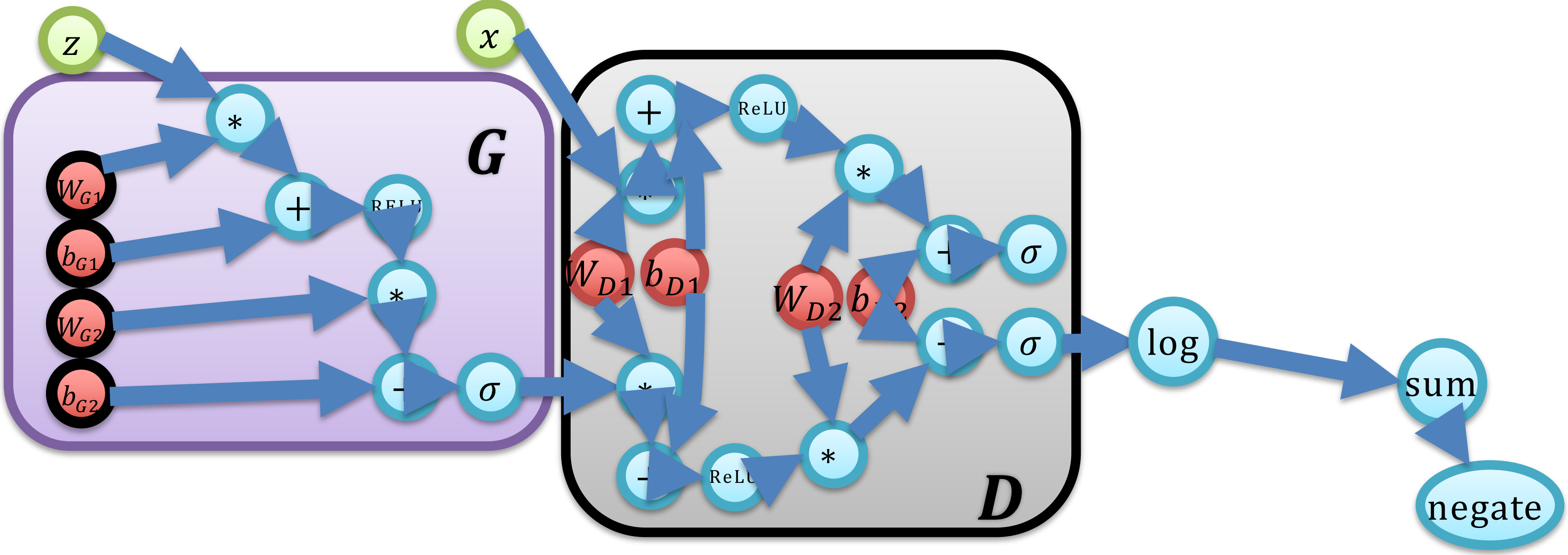
W_{D1}= tf.Variable(tf.truncated_normal([784, 256], stddev=0.1)



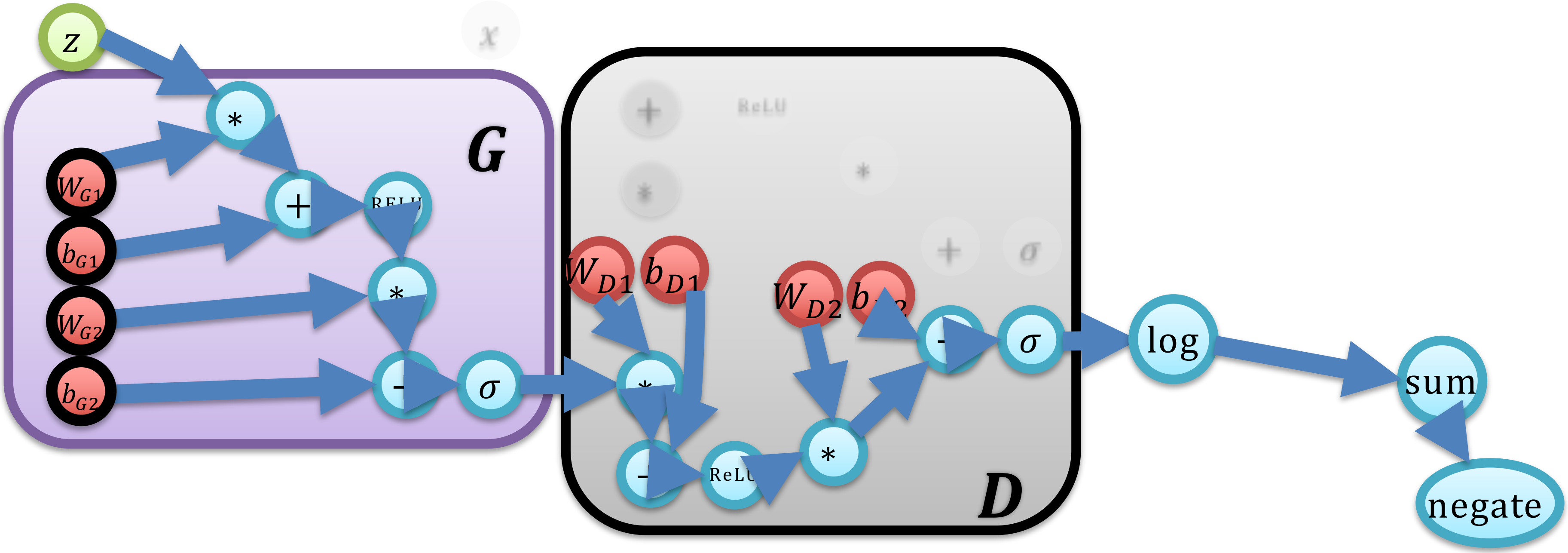

z
x
G
D
*
+
RELU
σ
W_{G1}
b_{G1}
W_{G2}
b_{G2}
W_{D1}
b_{D1}
W_{D2}
b_{D2}
ReLU



Loss when training D



Loss when training G



Loss when training G