CS/DS 552: Class 7

Jacob Whitehill

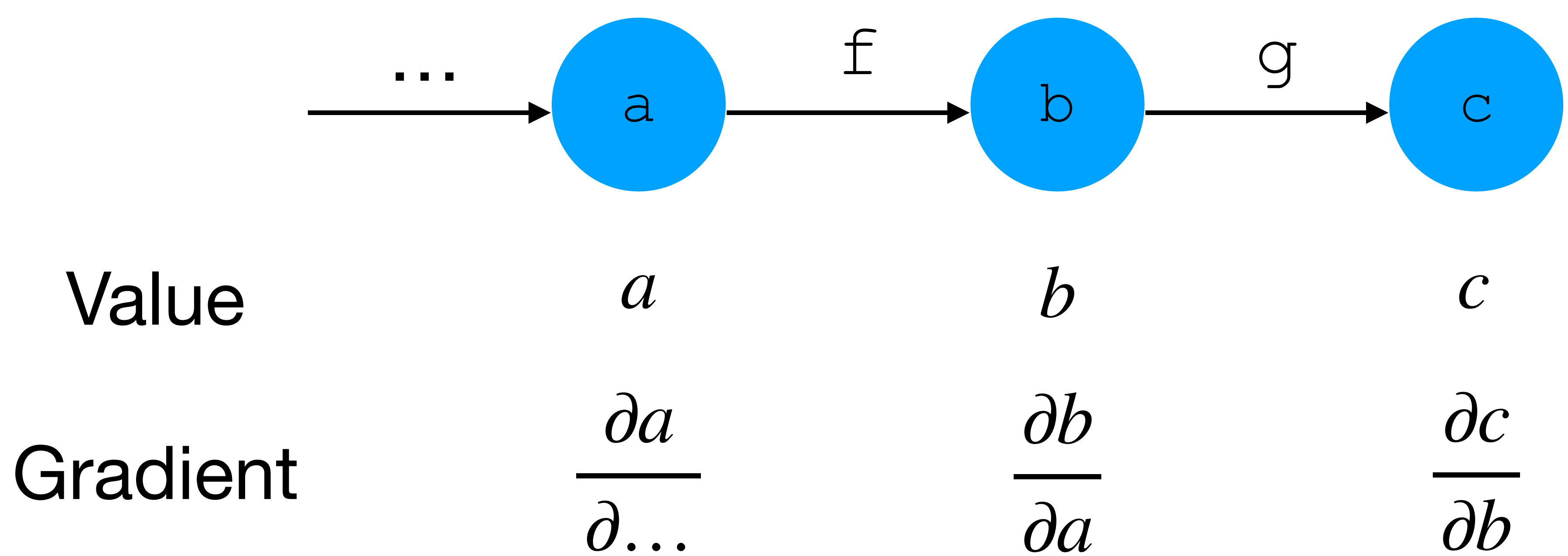
Controlling gradient flow in PyTorch

Controlling gradient flow in PyTorch

- Consider:

```
b = f(a)
c = g(b)
```

- Here, we can back-prop from `c` to `a` with $\dfrac{\partial c}{\partial a} = \dfrac{\partial c}{\partial b}\dfrac{\partial b}{\partial a}$.



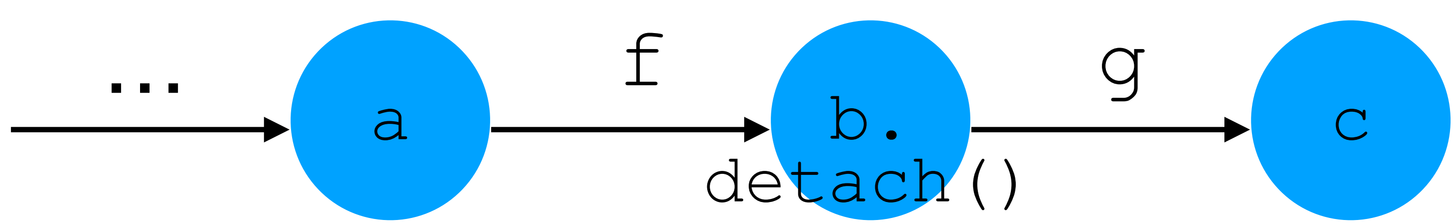
Controlling gradient flow in PyTorch

- But what if we call `b.detach()`:

```
b = f(a)
c = g(b.detach())
```

- We can still forward-prop from `a` to `c`.

- However, since $\dfrac{\partial b}{\partial a} = 0$ (according to autograd), we cannot back-prop through `f` to compute $\dfrac{\partial c}{\partial a}$.



Value	a	b	c
Gradient	$\dfrac{\partial a}{\partial \ldots}$	0	$\dfrac{\partial c}{\partial b}$

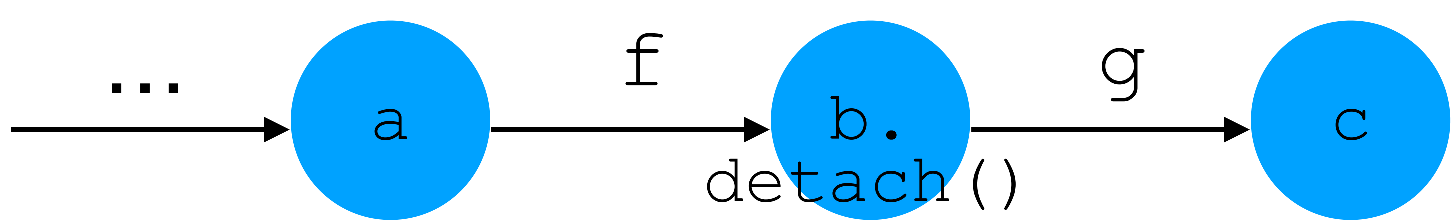
Controlling gradient flow in PyTorch

- But what if we call `b.detach()`:

```
b = f(a)
c = g(b.detach())
```

- We can still forward-prop from `a` to `c`.

- Hence, any parameters in `f` (or earlier in the graph) will *not* get updated in SGD.



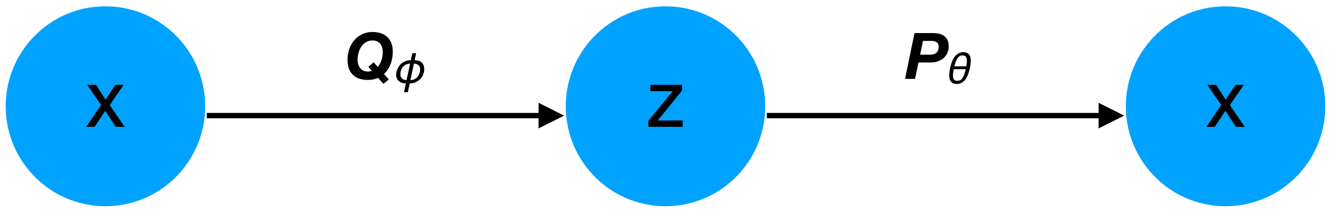
Value	a	b	c
Gradient	$\frac{\partial a}{\partial \ldots}$	0	$\frac{\partial c}{\partial b}$

Generative Adversarial Networks (GANs)

Generative models

- So far, the generative models we have discussed were latent variable models (LVMs).
- In these cases, we can train the model as the combination of an encoder Q and decoder P with a **single optimization objective** of maximizing the ELBO:

$$-D_{\mathrm{KL}}(Q_\phi(z \mid \mathbf{x}) \mid P(z)) + \mathbb{E}_{Q_\phi}[\log P(\mathbf{x} \mid z)]$$



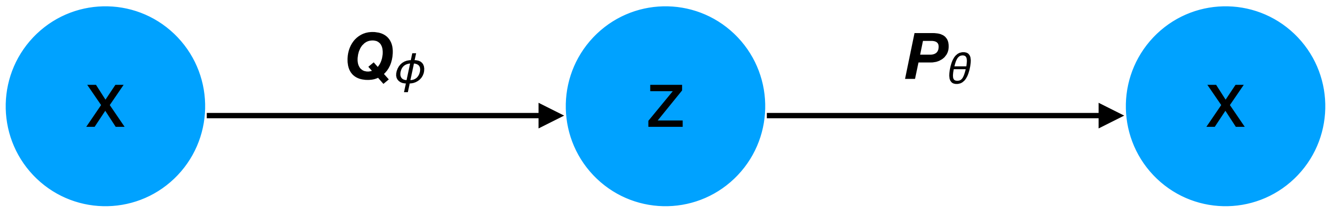
Generative models

- So far, the generative models we have discussed were latent variable models (LVMs).

- In these cases, we can train the model as the combination of an encoder *Q* and decoder *P* with a **single optimization objective** of maximizing the ELBO:

$$-D_{\mathrm{KL}}(Q_\phi(z \mid \mathbf{x}) \mid P(z)) + \mathbb{E}_{Q_\phi}[\log P(\mathbf{x} \mid z)]$$

- In other words, the encoder and decoder are working **cooperatively** to maximize the data log-likelihood.



Generative Adverarial Networks (GANs)

- However, another entire class of deep learning methods is based on training two networks that **compete against each other** in a zero-sum game.

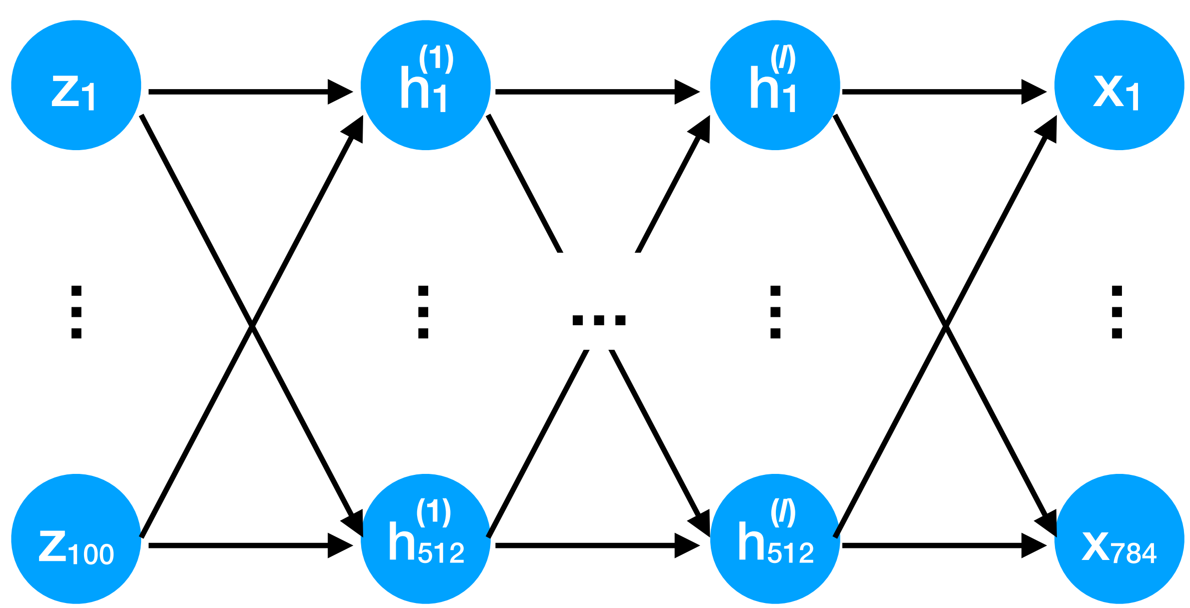
- This idea is the basis for the **Generative Adversarial Network** (**GAN**; Goodfellow et al. 2014).

Generative Adverarial Networks (GANs)

- Like VAEs, GANs consist of two components, but their semantics are different.
- Let $P_{\text{data}}(\mathbf{x})$ be the ground-truth data distribution.
- **Generator *G***: given a noise vector **z** from an easy-to-sample distribution (e.g., Gaussian, uniform), generate a vector **x** that looks like it came from $P_{\text{data}}(\mathbf{x})$.
- **Discriminator *D***: given a vector **x**, decide if it is real ($\hat{y}$=1) or fake ($\hat{y}$=0). *D* acts as a “forgery detector”.

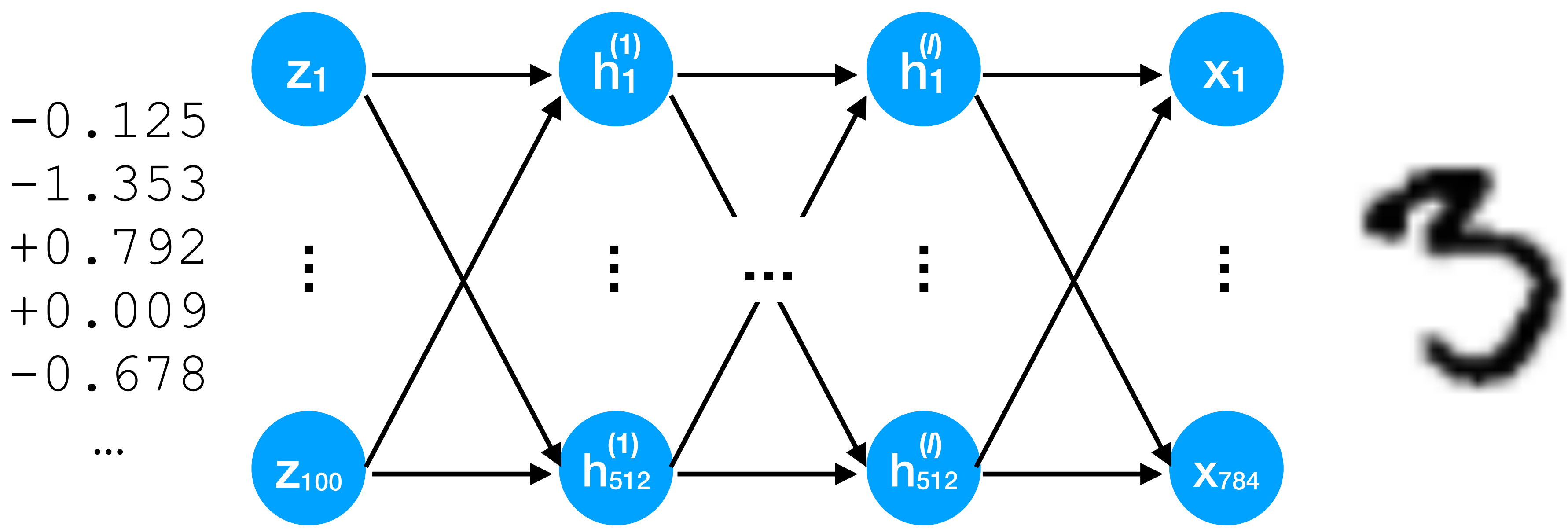
Generator *G*

- Example *G* with *l* hidden layers that generates an MNIST image (28x28=784) **x** from a 100-dim noise vector **z:**



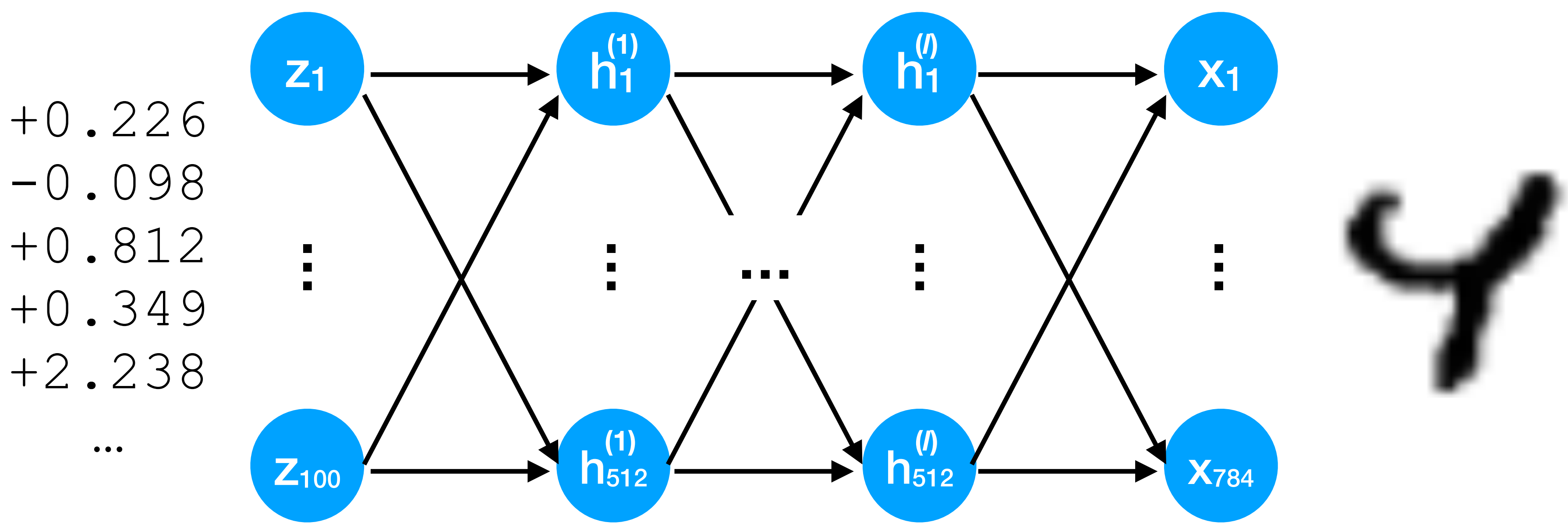
Generator G

- By feeding different noise vectors **z**, we obtain different **x**.
- Implicitly, **z** encodes the different dimensions of variability of $P_{\text{data}}(\mathbf{x})$ (though they may not be intuitive, independent, or **disentangled**).



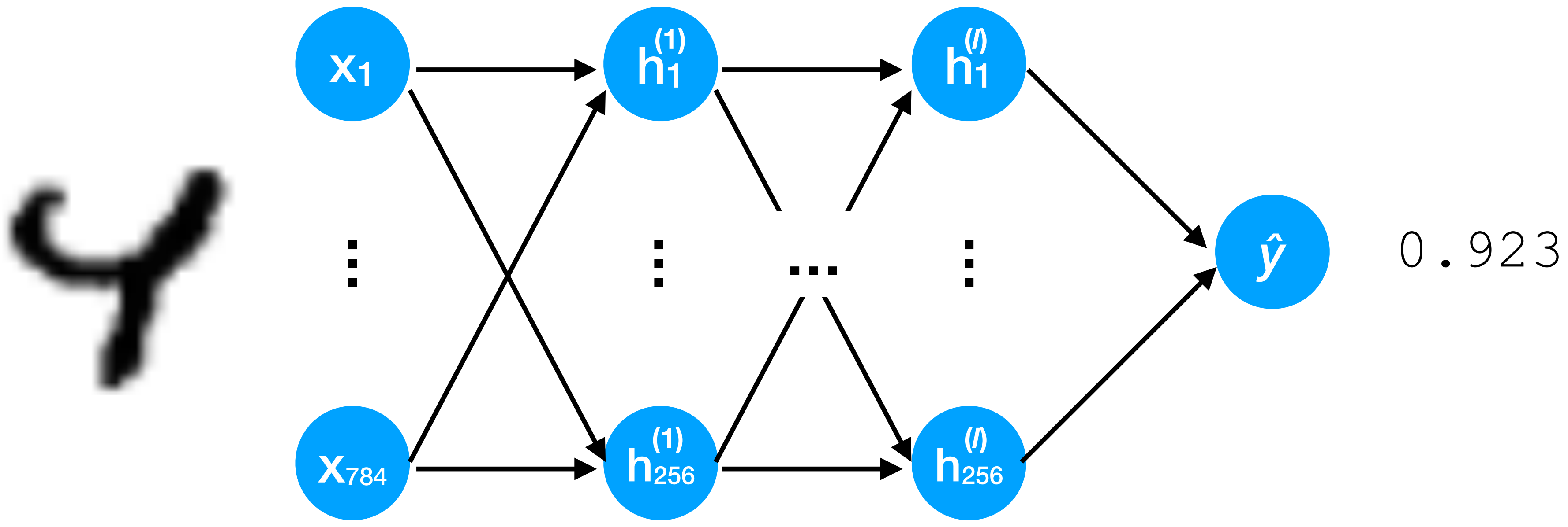
Generator G

- By feeding different noise vectors **z**, we obtain different **x**.
- Implicitly, **z** encodes the different dimensions of variability of $P_{\text{data}}(\mathbf{x})$ (though they may not be intuitive, independent, or **disentangled**).



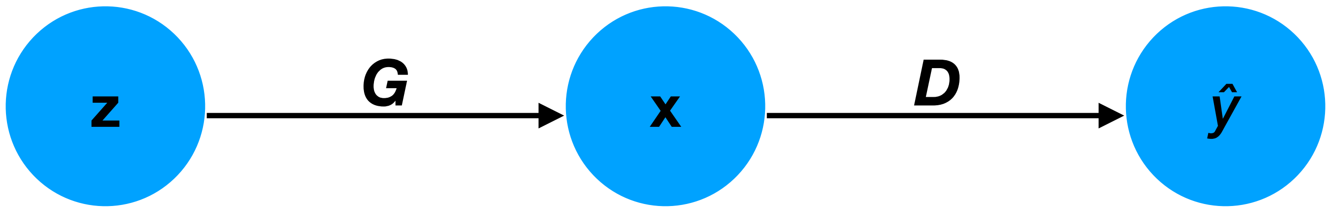
Discriminator *D*

- Example *D* with *l* hidden layers that estimates $\hat{y} \in (0,1)$ that expresses probability that the input **x** is real:



Generative Adverarial Networks (GANs)

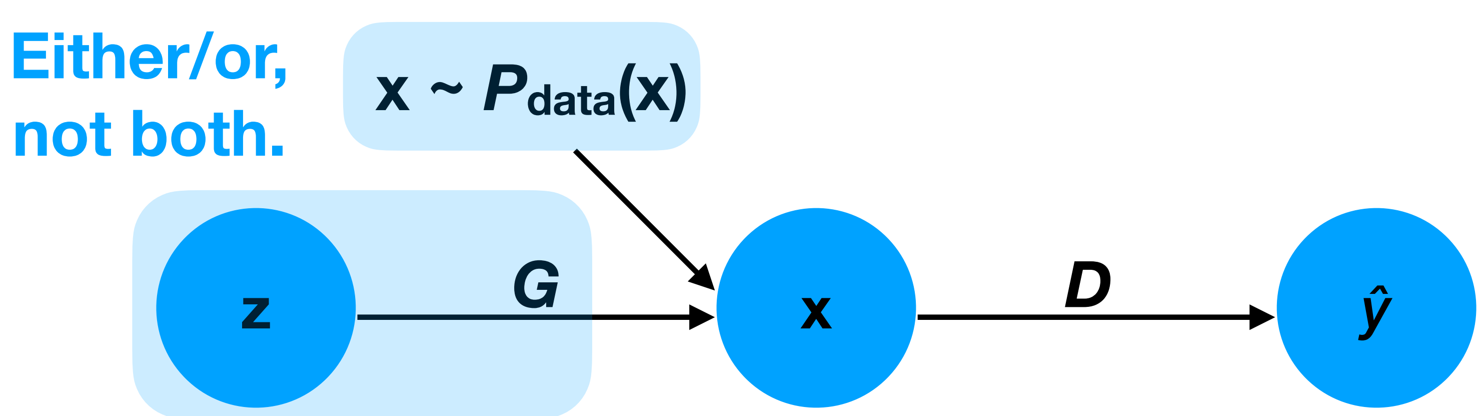
- Like VAEs, GANs are trained such that one component "feeds" to the other.



Generative Adverarial Networks (GANs)

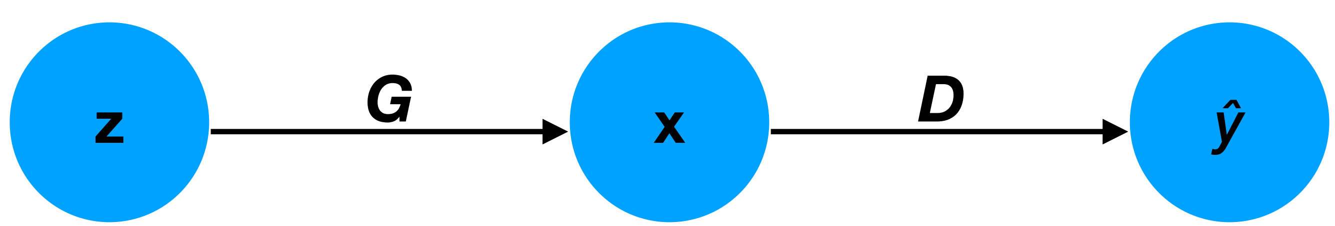
- Like VAEs, GANs are trained such that one component “feeds” to the other.

- In contrast to VAEs, the discriminator *D* is sometimes given a “fake” data vector **x** (generated by *G*), and sometimes given a “real” vector x sampled from the training set (which approximates $P_{\text{data}}(\mathbf{x})$).



Generative Adverarial Networks (GANs)

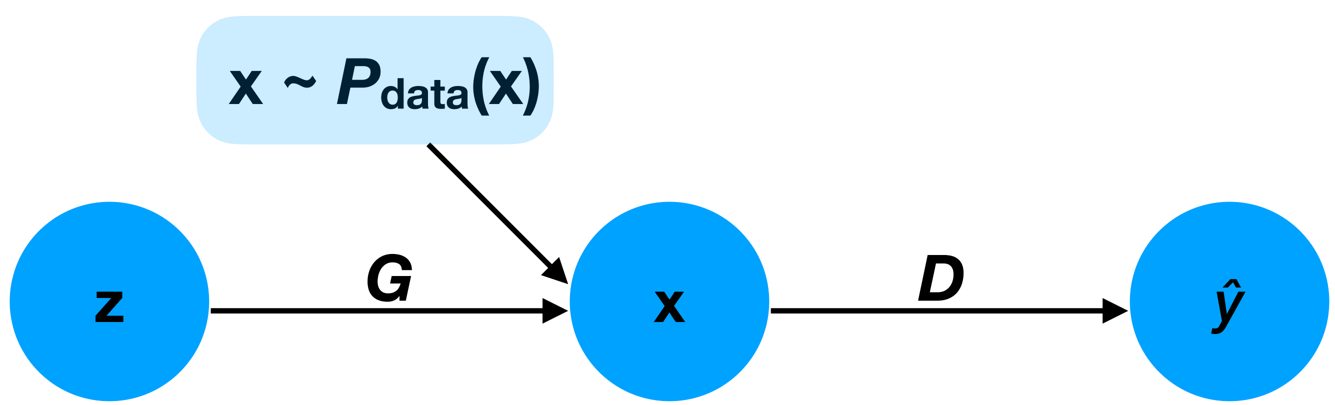
- Each network has its own parameters:
  - G has parameters θ_G.
  - D has parameters θ_D.



Generative Adverarial Networks (GANs)

- We can define the following accuracy function of how well *D* can discriminate fake from real data:

$$f_{\text{acc}}(\theta_G, \theta_D) = \underbrace{\mathbb{E}_{\mathbf{x}\sim P_{\text{data}}(\mathbf{x})}[\log D_{\theta_D}(\mathbf{x})]}_{\text{Log-likelihood that } D \text{ recognizes real data as real.}} + \mathbb{E}_{\mathbf{z}\sim P(\mathbf{z})}[\log(1 - D_{\theta_D}(G_{\theta_G}(\mathbf{z})))]$$

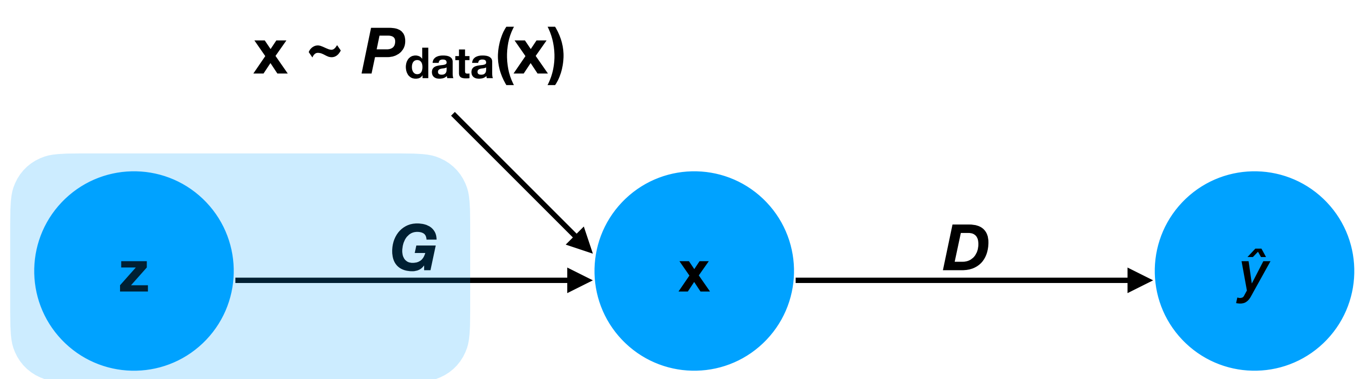


Generative Adverarial Networks (GANs)

- We can define the following accuracy function of how well *D* can discriminate fake from real data:

$$f_{\text{acc}}(\theta_G, \theta_D) = \mathbb{E}_{\mathbf{x}\sim P_{\text{data}}(\mathbf{x})}[\log D_{\theta_D}(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim P(\mathbf{z})}[\log(1 - D_{\theta_D}(G_{\theta_G}(\mathbf{z})))]$$

Log-likelihood that *D* recognizes fake data as fake.



Generative Adverarial Networks (GANs)

- The goal of *D* is to *maximize* f_{acc}, whereas the goal of *G* is to *minimize* f_{acc}.

$$f_{\text{acc}}(\theta_G, \theta_D) = \mathbb{E}_{\mathbf{x} \sim P_{\text{data}}(\mathbf{x})}[\log D_{\theta_D}(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim P(\mathbf{z})}[\log(1 - D_{\theta_D}(G_{\theta_G}(\mathbf{z})))]$$

- *D* and *G* compete against each other to find equilibrium:

$$\min_{\theta_G} \max_{\theta_D} f_{\text{acc}}(\theta_G, \theta_D)$$

- In particular, this solution corresponds to *D* having 50% accuracy at detecting forgeries, and *G* generating fake **x** according to $P_{\text{data}}(\mathbf{x})$.

Training GANs

$$f_{\text{acc}}(\theta_G, \theta_D) = \mathbb{E}_{\mathbf{x} \sim P_{\text{data}}(\mathbf{x})}[\log D_{\theta_D}(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim P(\mathbf{z})}[\log(1 - D_{\theta_D}(G_{\theta_G}(\mathbf{z})))]$$

- In practice, we train *D* and *G* *iteratively*:
  - Freeze *G*, and perform SGD on *D* for *k* iterations to increase f_{acc}.

Training GANs

$$f_{\text{acc}}(\theta_G, \theta_D) = \mathbb{E}_{\mathbf{x}\sim P_{\text{data}}(\mathbf{x})}[\log D_{\theta_D}(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim P(\mathbf{z})}[\log(1 - D_{\theta_D}(G_{\theta_G}(\mathbf{z})))]$$

- In practice, we train *D* and *G* *iteratively*:
    - Freeze *G*, and perform SGD on *D* for *k* iterations to increase f_{acc}.

Improve *D*'s forgery detection accuracy for a fixed distribution of fake data.

Training GANs

$$f_{\text{acc}}(\theta_G, \theta_D) = \mathbb{E}_{\mathbf{x}\sim P_{\text{data}}(\mathbf{x})}[\log D_{\theta_D}(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim P(\mathbf{z})}[\log(1 - D_{\theta_D}(G_{\theta_G}(\mathbf{z})))]$$

- In practice, we train *D* and *G* *iteratively*:
  - Freeze *G*, and perform SGD on *D* for *k* iterations to increase f_{acc}.
  - Freeze *D*, and perform SGD on *G* for *l* iterations to decrease f_{acc}.

Training GANs

$$f_{\text{acc}}(\theta_G, \theta_D) = \mathbb{E}_{\mathbf{x} \sim P_{\text{data}}(\mathbf{x})}[\log D_{\theta_D}(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim P(\mathbf{z})}[\log(1 - D_{\theta_D}(G_{\theta_G}(\mathbf{z})))]$$

- In practice, we train *D* and *G* *iteratively*:
  - Freeze *G*, and perform SGD on *D* for *k* iterations to increase f_{acc}.
  - Freeze *D*, and perform SGD on *G* for *l* iterations to decrease f_{acc}.

Improve *G* for a fixed forgery detector *D*.

Difficulty in training

- GANs are renowned for being difficult to train:

  1. How to choose *k*, *l*? More hyperparameters to optimize.

Difficulty in training

- GANs are renowned for being difficult to train:

  1. How to choose k, l? More hyperparameters to optimize.

  2. We will probably never reach the equilibrium where G exactly produces $P_{\text{data}}(\mathbf{x})$ and D's accuracy is 0.5.
     - What kind of “training curve” for D, G should we expect?
     - If D gets too good too fast, then G may never have a chance to improve.

Difficulty in training

- GANs are renowned for being difficult to train:

  3. **Mode collapse** — *G* generates realistic data but only for a *subset* of the domain of $P_{\text{data}}(\mathbf{x})$, e.g.:

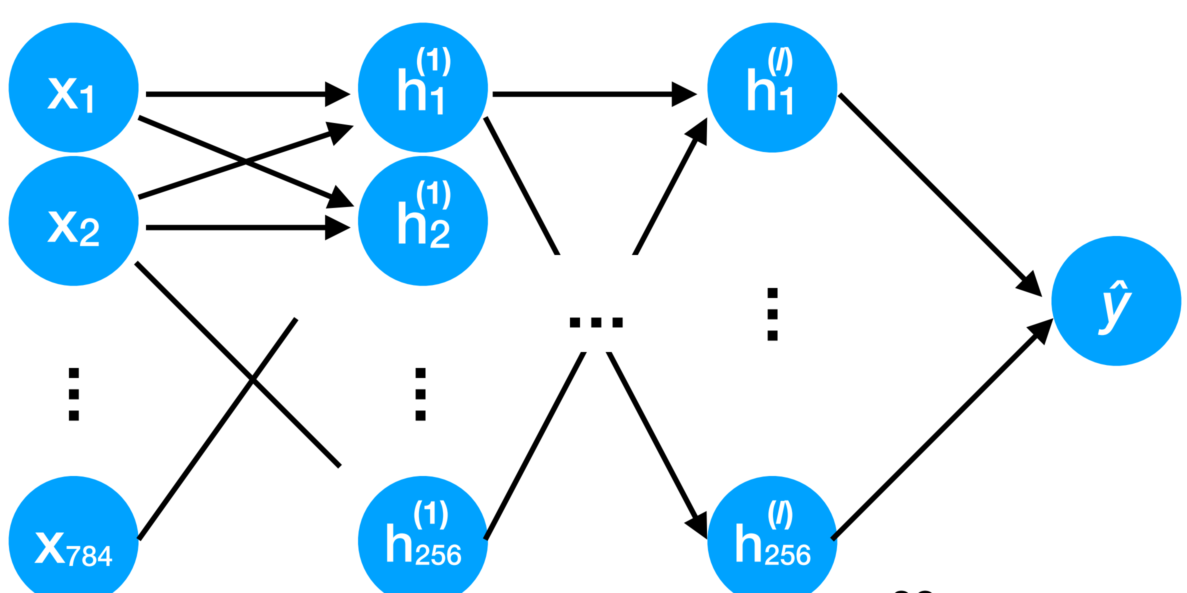
https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/

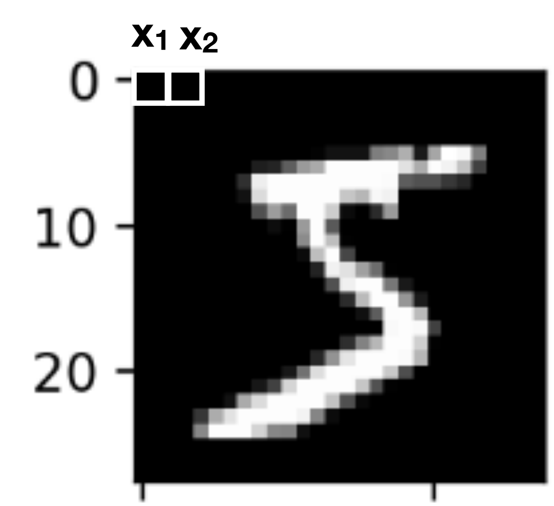
Difficulty in training

- GANs are renowned for being difficult to train:

  3. **Neuron co-adaptation** — training gets stuck because multiple NN pathways rely on each other too much.

     - Consider an MNIST image near the borders: what property do pixels x_1, x_2 have?
       - $x_1 = x_2 = 0$?



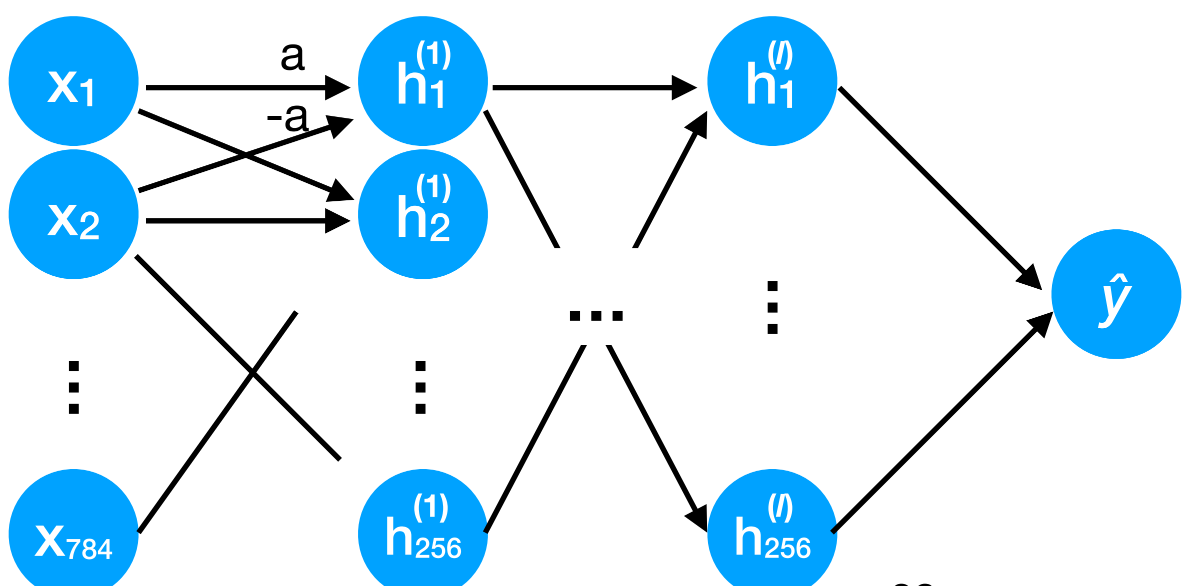


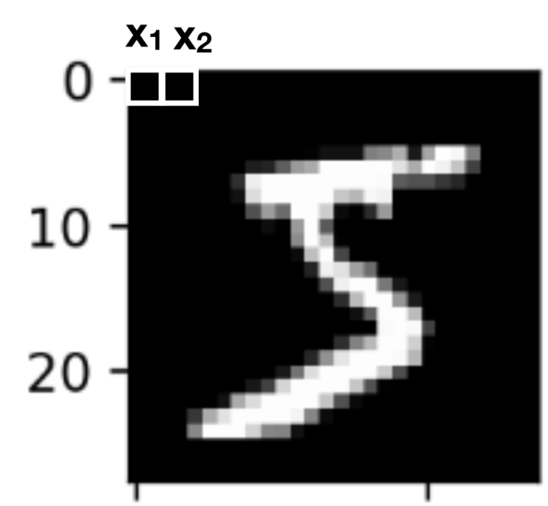
https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/

Difficulty in training

- GANs are renowned for being difficult to train:

  3. **Neuron co-adaptation** — training gets stuck because multiple NN pathways rely on each other too much.
     - Consider an MNIST image near the borders: what property do pixels x_1, x_2 have?
       - $ax_1 - ax_2 = 0$?





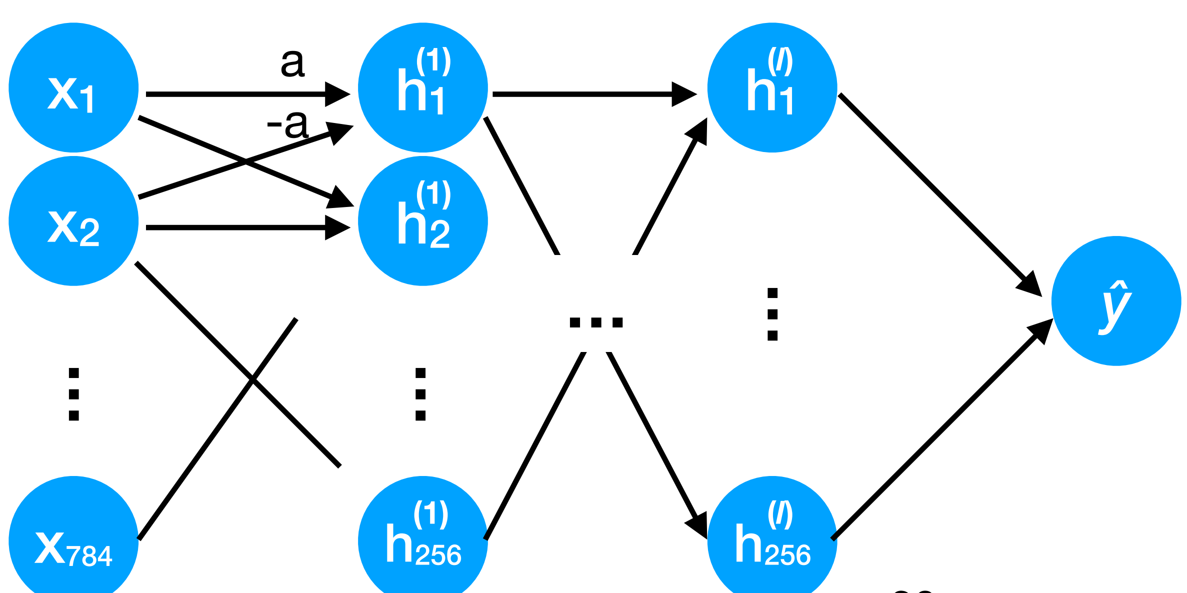
https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/

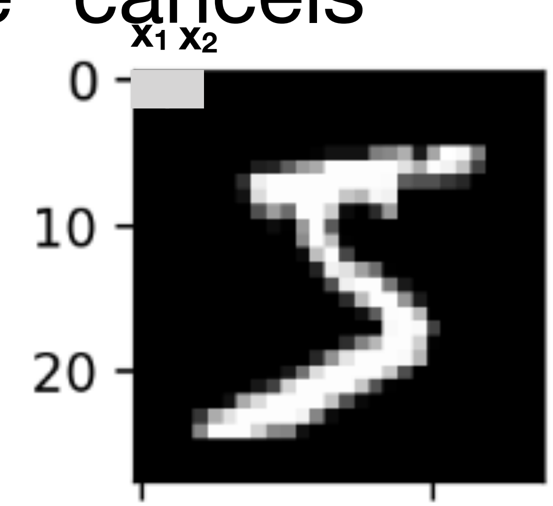
Difficulty in training

- GANs are renowned for being difficult to train:

  3. **Neuron co-adaptation** — training gets stuck because multiple NN pathways rely on each other too much.

     - In the latter case, *D* gives feedback to *G* that images are “ok" as long the background noise “cancels” itself, e.g.:





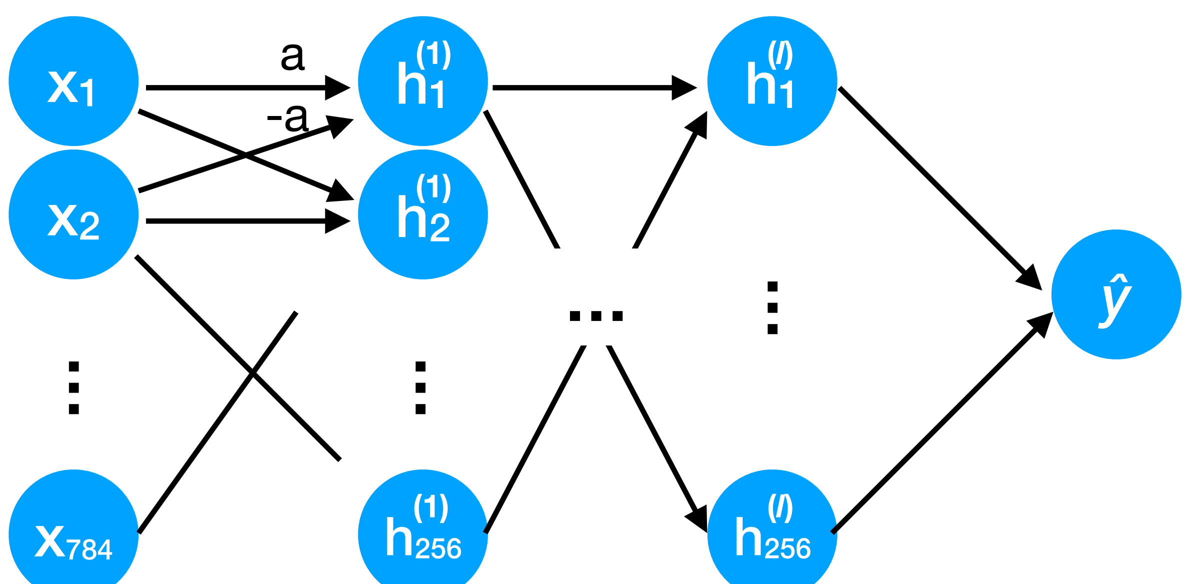
https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/

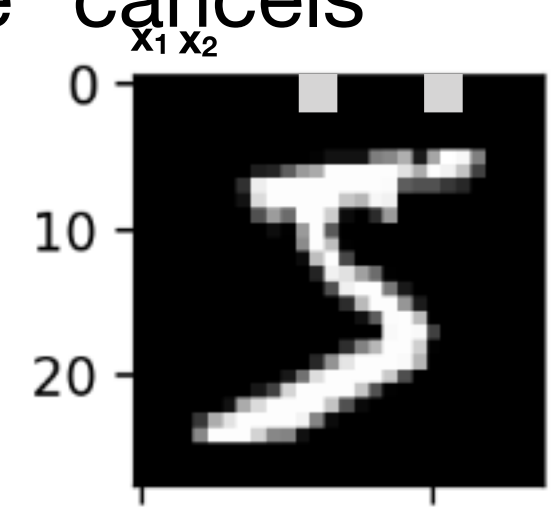
Difficulty in training

- GANs are renowned for being difficult to train:

  3. **Neuron co-adaptation** — training gets stuck because multiple NN pathways rely on each other too much.

     - In the latter case, *D* gives feedback to *G* that images are “ok" as long the background noise “cancels” itself, e.g.:





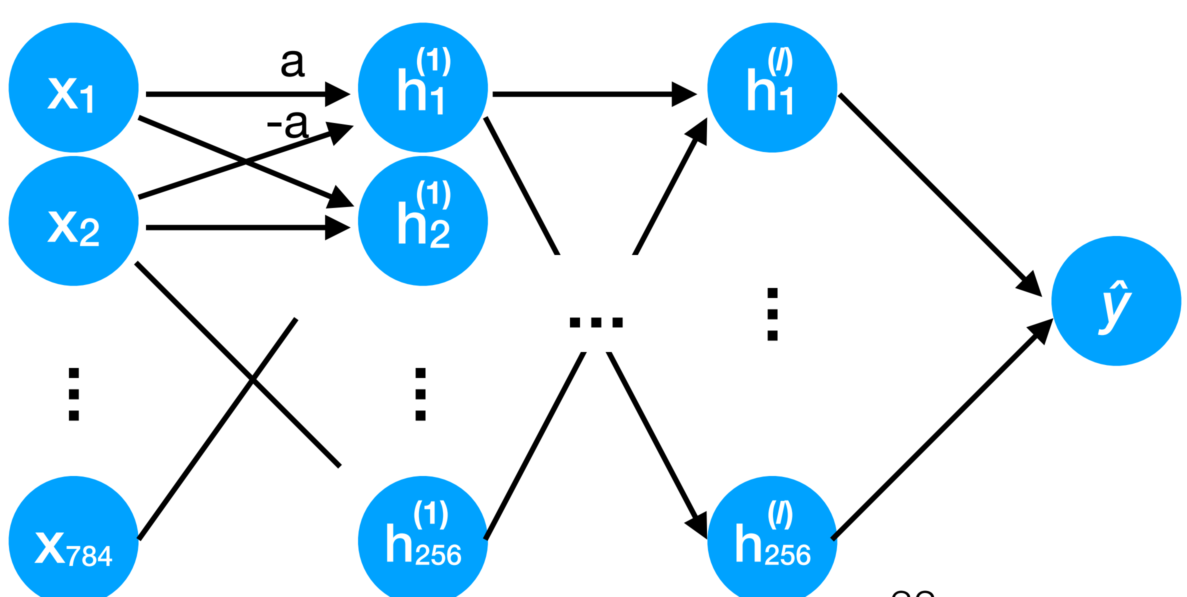
https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/

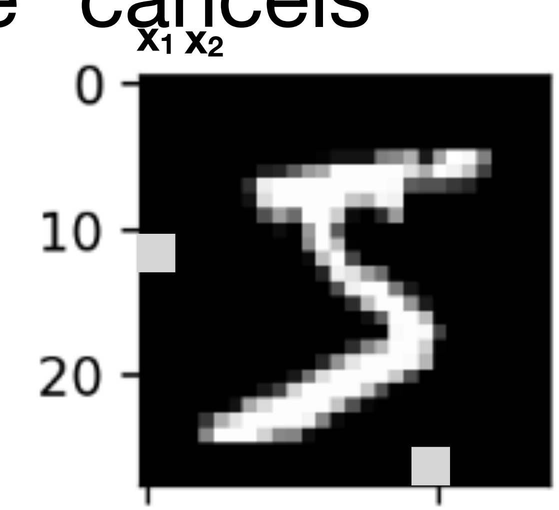
Difficulty in training

- GANs are renowned for being difficult to train:

  3. **Neuron co-adaptation** — training gets stuck because multiple NN pathways rely on each other too much.

     - In the latter case, *D* gives feedback to *G* that images are “ok" as long the background noise “cancels” itself, e.g.:





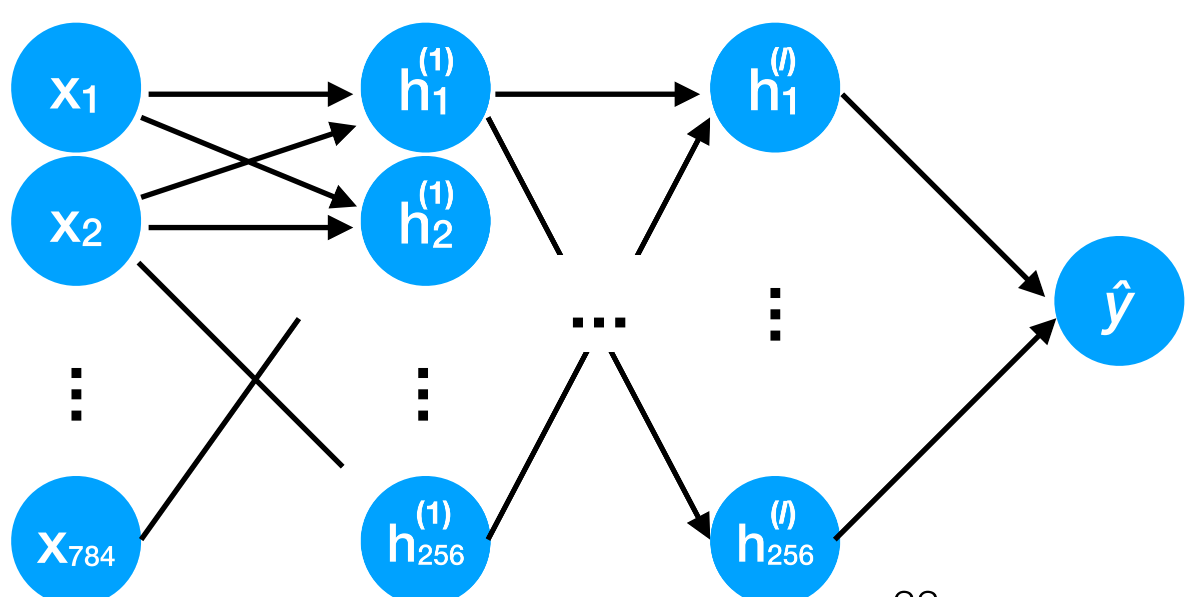
https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/

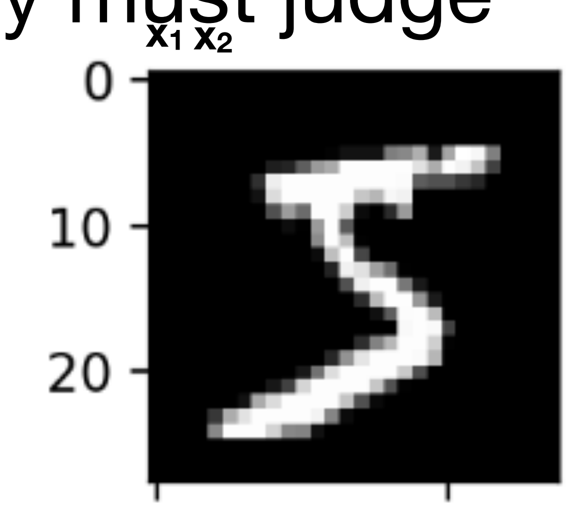
Difficulty in training

- GANs are renowned for being difficult to train:

  3. **Neuron co-adaptation** — training gets stuck because multiple NN pathways rely on each other too much.

     - To prevent this from occurring, we can use dropout on the input layer **x**, so that each pathway must judge independently if the image is a fake.





https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/

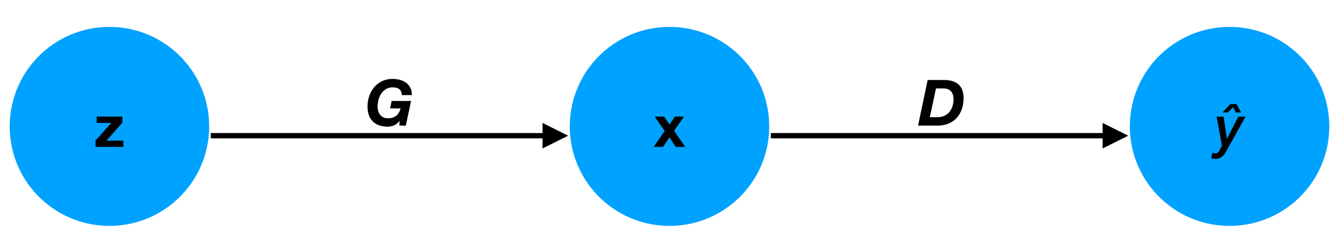
Closer look at f_{acc}

- Consider the loss term for fake data:

$$f_{\text{acc}}(\theta_G, \theta_D) = \mathbb{E}_{\mathbf{x}\sim P_{\text{data}}(\mathbf{x})}[\log D_{\theta_D}(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim P(\mathbf{z})}[\log(1 - D_{\theta_D}(G_{\theta_G}(\mathbf{z})))]$$

- What happens early during training, when *G* is not very good (but *D* typically is fairly good)?

$$\nabla_{\theta_G} \log(1 - D(\mathbf{x})) = -\frac{1}{1 - D(\mathbf{x})} \frac{\partial D}{\partial \mathbf{x}} \frac{\partial \mathbf{x}}{\partial \theta_G}$$



Closer look at f_{acc}

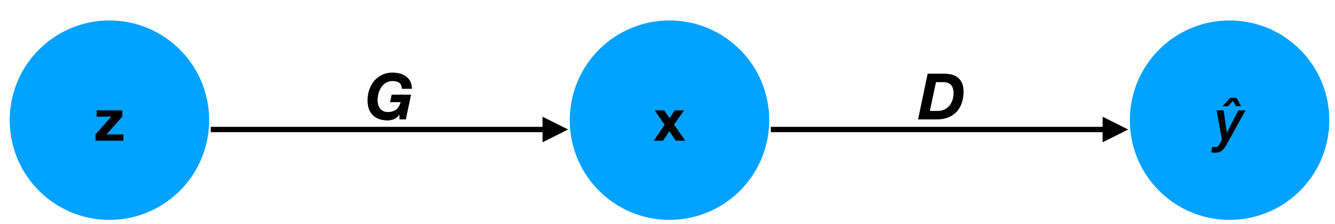
- Consider the loss term for fake data:

$$f_{\text{acc}}(\theta_G, \theta_D) = \mathbb{E}_{\mathbf{x}\sim P_{\text{data}}(\mathbf{x})}[\log D_{\theta_D}(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim P(\mathbf{z})}[\log(1 - D_{\theta_D}(G_{\theta_G}(\mathbf{z})))]$$

- What happens early during training, when *G* is not very good (but *D* typically is fairly good)?

- *D* ≈ 0 and d*D*/d**x** ≈ 0.

$$\begin{aligned}\nabla_{\theta_G} \log(1 - D(\mathbf{x})) &= -\frac{1}{1 - D(\mathbf{x})} \frac{\partial D}{\partial \mathbf{x}} \frac{\partial \mathbf{x}}{\partial \theta_G} \\ &\approx -1 \frac{\partial D}{\partial \mathbf{x}} \frac{\partial \mathbf{x}}{\partial \theta_G}\end{aligned}$$



Closer look at f_{acc}

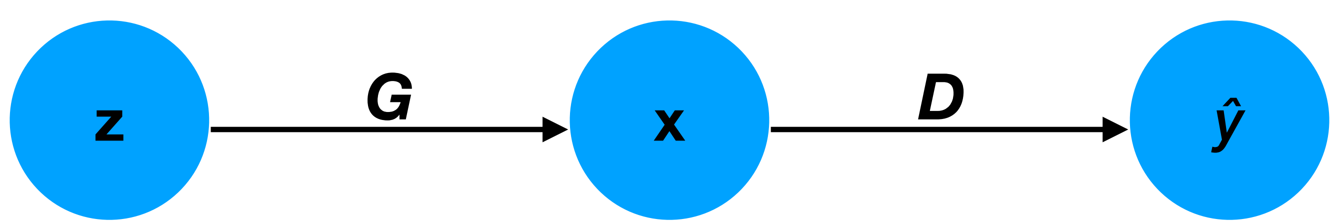
- Consider the loss term for fake data:

$$f_{\text{acc}}(\theta_G, \theta_D) = \mathbb{E}_{\mathbf{x}\sim P_{\text{data}}(\mathbf{x})}[\log D_{\theta_D}(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim P(\mathbf{z})}[\log(1 - D_{\theta_D}(G_{\theta_G}(\mathbf{z})))]$$

- What happens early during training, when *G* is not very good (but *D* typically is fairly good)?

- $D \approx 0$ and $dD/d\mathbf{x} \approx 0$.

$$\begin{aligned} \nabla_{\theta_G} \log(1 - D(\mathbf{x})) &= -\frac{1}{1 - D(\mathbf{x})} \frac{\partial D}{\partial \mathbf{x}} \frac{\partial \mathbf{x}}{\partial \theta_G} \\ &\approx -1 \frac{\partial D}{\partial \mathbf{x}} \frac{\partial \mathbf{x}}{\partial \theta_G} \\ &\approx -1 \times \text{small} \times \frac{\partial \mathbf{x}}{\partial \theta_G} \end{aligned}$$



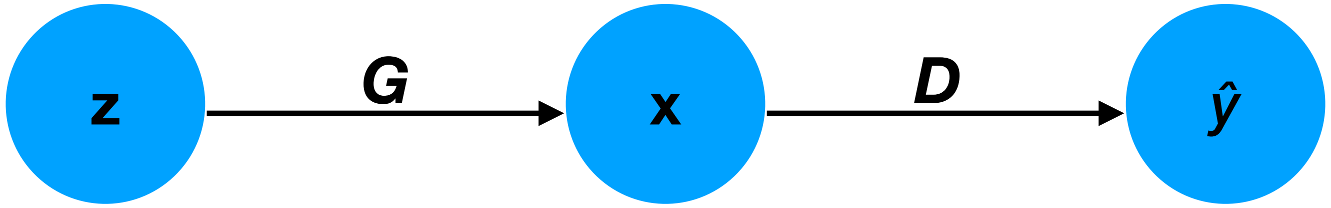
Closer look at f_{acc}

- To accelerate training early on, we can instead use a different loss term for the fake data that yields the same desired behavior but trains faster.

$$f_{\text{acc}}(\theta_G, \theta_D) = \mathbb{E}_{\mathbf{x} \sim P_{\text{data}}(\mathbf{x})}[\log D_{\theta_D}(\mathbf{x})] + \boxed{\mathbb{E}_{\mathbf{z} \sim P(\mathbf{z})}[-\log(D_{\theta_D}(G_{\theta_G}(\mathbf{z})))]}$$

New loss term

$$\nabla_{\theta_G} - \log D(\mathbf{x}) = -\frac{1}{D(\mathbf{x})} \frac{\partial D}{\partial \mathbf{x}} \frac{\partial \mathbf{x}}{\partial \theta_G}$$
$$\approx -\frac{1}{\text{small}} \times \text{small} \times \frac{\partial \mathbf{x}}{\partial \theta_G}$$



Closer look at f_{acc}

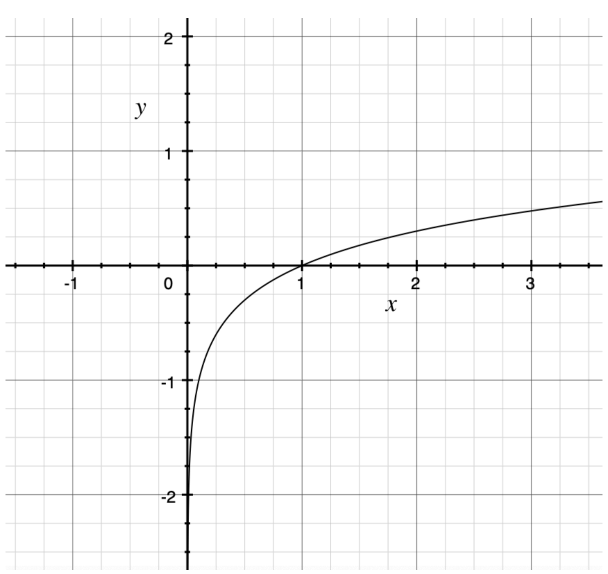
- To accelerate training early on, we can instead use a different loss term for the fake data that yields the same desired behavior but trains faster.

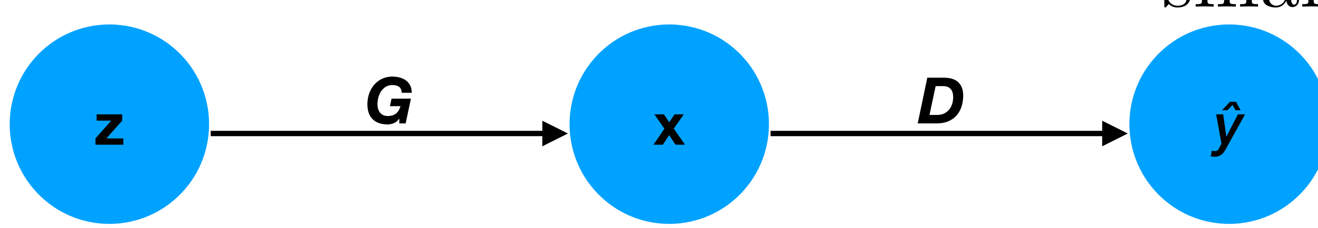
$$f_{\text{acc}}(\theta_G, \theta_D) = \mathbb{E}_{\mathbf{x}\sim P_{\text{data}}(\mathbf{x})}[\log D_{\theta_D}(\mathbf{x})] + \underbrace{\mathbb{E}_{\mathbf{z}\sim P(\mathbf{z})}[-\log(D_{\theta_D}(G_{\theta_G}(\mathbf{z})))]}_{\text{New loss term}}$$

- The reason is that the gradient of -log(*q*) for q≈0 is very large, whereas the gradient of log(1-*q*)≈1.

$$\nabla_{\theta_G} - \log D(\mathbf{x}) = -\frac{1}{D(\mathbf{x})}\frac{\partial D}{\partial \mathbf{x}}\frac{\partial \mathbf{x}}{\partial \theta_G}$$

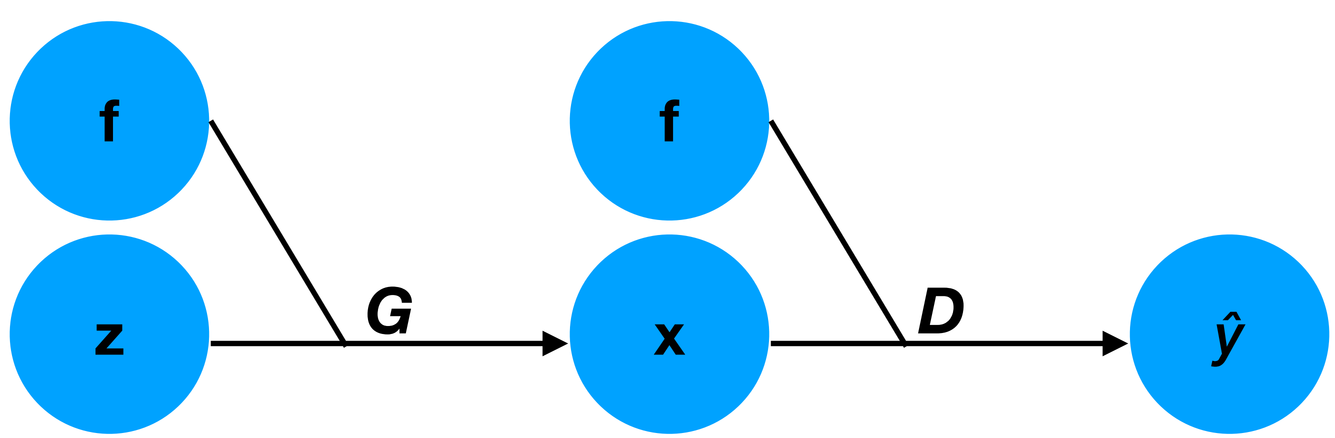
$$\approx -\frac{1}{\text{small}} \times \text{small} \times \frac{\partial \mathbf{x}}{\partial \theta_G}$$





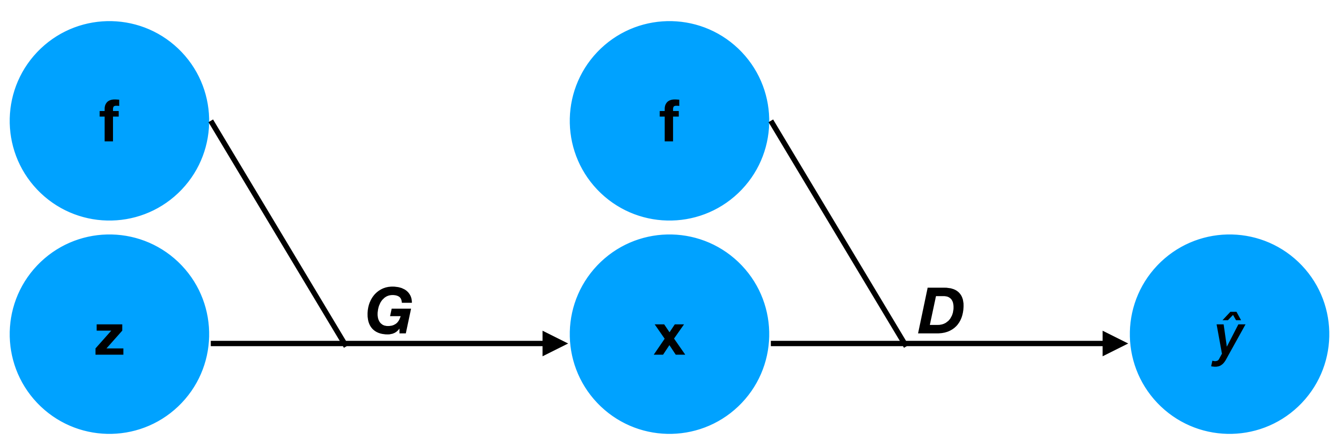
Conditional GANs

- One GAN variant is a **conditional GAN (c-GAN):**
    - *G* also accepts a feature vector **f** (e.g., 1-hot encoding of MNIST class) that specifies what *kind* of data to generate.
    - *D* also accepts **f** to help discriminate a particular kind of real from fake data.



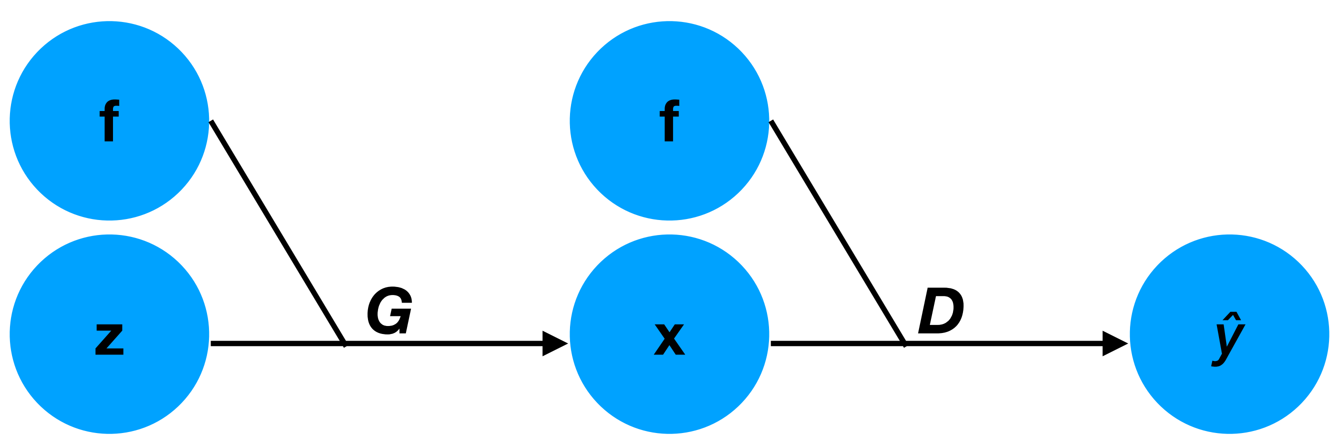
Conditional GANs

- Why is **f** necessary in *G*?
- Why is **f** necessary in *D*?
- Why better than just training a separate GAN for each **f**?



Conditional GANs

- Why is **f** necessary in *G*?
  - Need to know what class to generate.
- Why is **f** necessary in *D*?
  - Need to know what class was generated — a realistic 3 is still a fake 7!
- Why better than just training a separate GAN for each **f**?
  - Share knowledge across many classes.



Exercise

- Original loss:

$$f_{\mathrm{acc}}(\theta_G, \theta_D) = \mathbb{E}_{\mathbf{x} \sim P_{\mathrm{data}}(\mathbf{x})}[\log D_{\theta_D}(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim P(\mathbf{z})}[\log(1 - D_{\theta_D}(G_{\theta_G}(\mathbf{z})))]$$

- Goodfellow’s modified loss:

$$f_{\mathrm{acc}}(\theta_G, \theta_D) = \mathbb{E}_{\mathbf{x} \sim P_{\mathrm{data}}(\mathbf{x})}[\log D_{\theta_D}(\mathbf{x})] + \mathbb{E}_{\mathbf{z} \sim P(\mathbf{z})}[-\log(D_{\theta_D}(G_{\theta_G}(\mathbf{z})))]$$

Don’t get too “attached” to the code.