CS/DS 541: Class 19

Jacob Whitehill

Exercises

https://cs230.stanford.edu/files/cs230exam_fall19_soln.pdf

Exercise 4

Consider a model trying to learn an encoding of some input $x \in \mathbb{R}$. The goal is to encode the input x using $z = w_1 x \in \mathbb{R}$, then accurately reconstruct the original x from the encoded representation using $\hat{x} = w_2 z \in \mathbb{R}$. Here, $(w_1, w_2) \in \mathbb{R} \times \mathbb{R}$. The model is trained with the squared reconstruction error:

$$L(W) = \frac{1}{n} \sum_{i=1}^{n} (x^{(i)} - w_2 w_1 x^{(i)})^2$$

(2 points) What is the set of solutions for w_1 and w_2 which makes loss zero?

(3 points) Does the loss have a saddle point? Where?

Generative Adversarial Networks (GANs)

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.

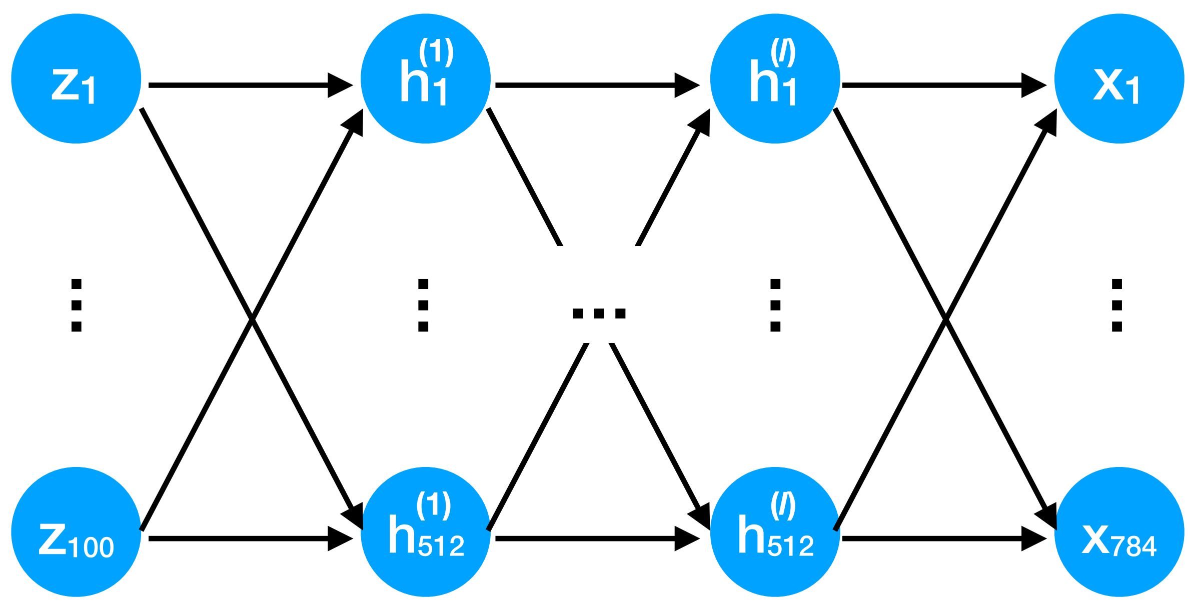
- In particular, the most prominent method (as of 2020) for generating novel data **x** is the Generative Adversarial Network (GAN; Goodfellow et al. 2014).

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”.

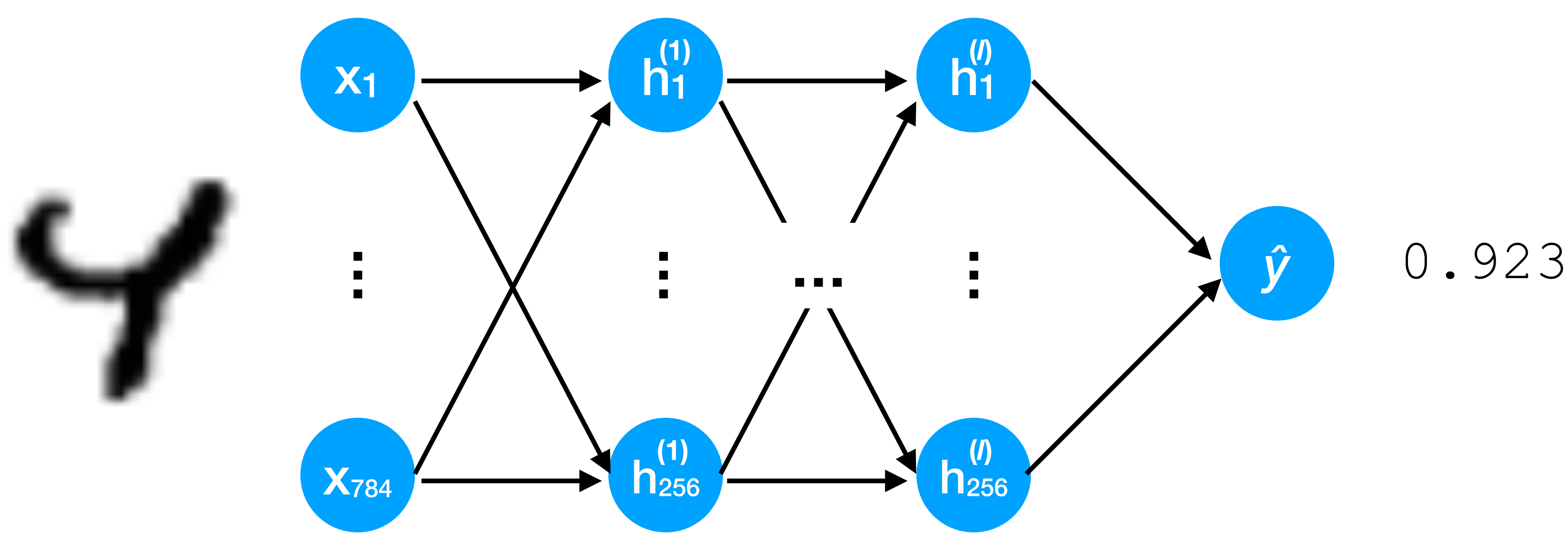
Generator *G*

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



Discriminator *D*

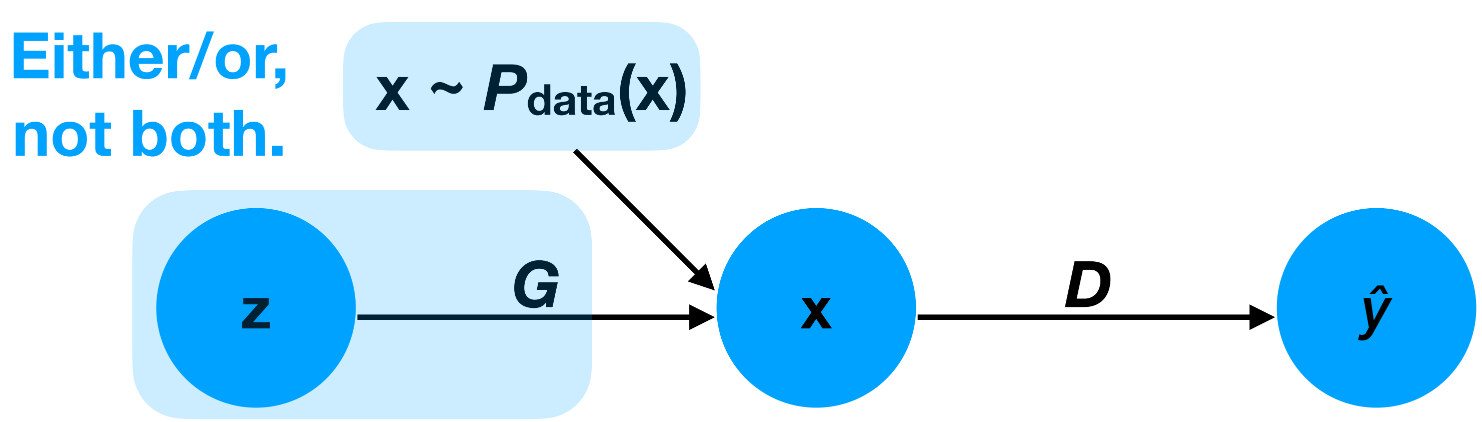
- Example *D* with *l* hidden layers that estimates $\hat{y} \in (0,1)$ that expresses probability that the input **x** is real:



Generative Adverarial Networks (GANs)

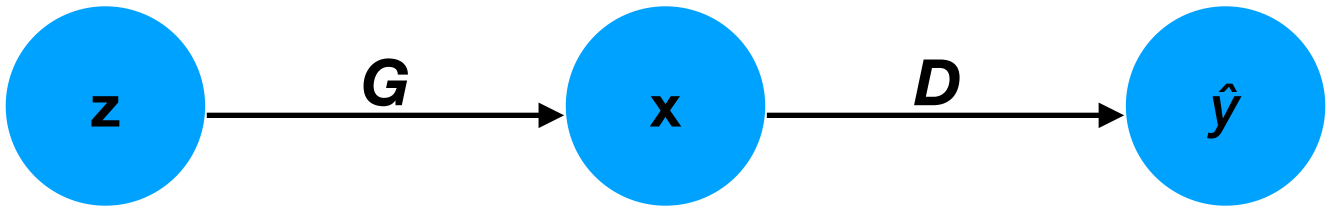
- Like VAEs, GANs are trained such that one component "feeds" to the other.

- In contrast to VAEs, the discriminator D is sometimes given a "fake" data vector $\mathbf{x}$ (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)

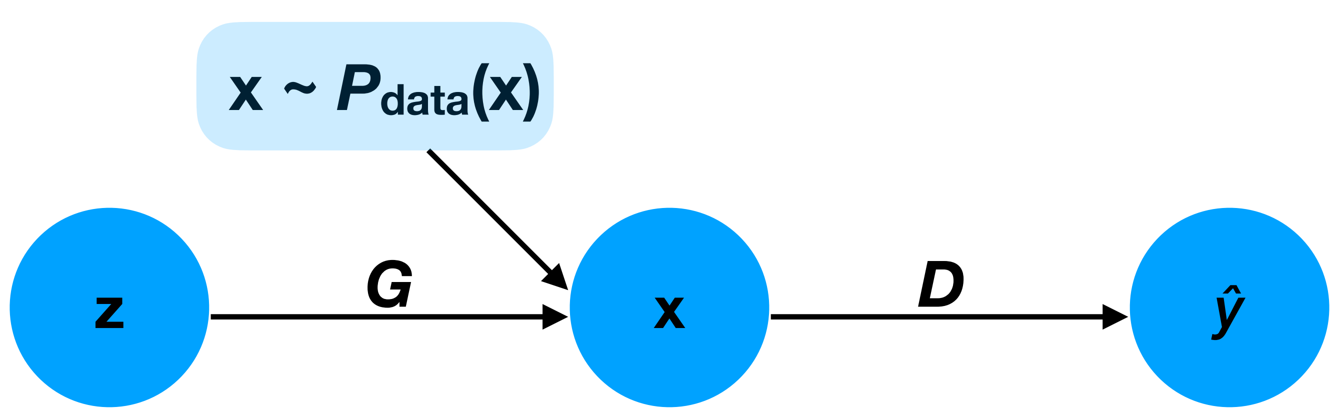
- Each network has its own parameters:
  - G has parameters θ_G.
  - D has parameters θ_D.



Generative Adverarial Networks (GANs)

- We can define the following loss on how well *D* can discriminate fake from real data:

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

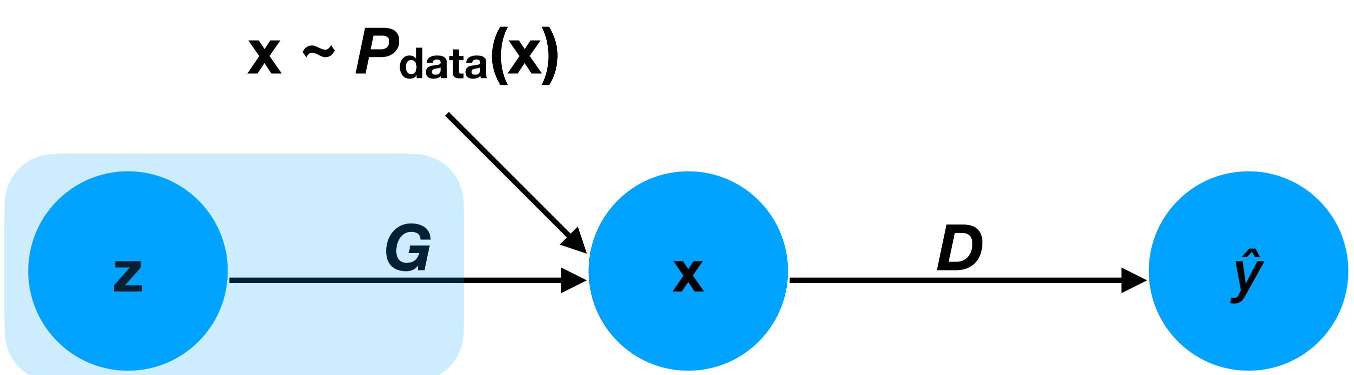


Generative Adverarial Networks (GANs)

- We can define the following loss on 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}.

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

- This two-player game will reach an equilibrium if we find:

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

- In particular, this solution corresponds to D having 50% accuracy at detecting forgeries, and G generating fake $\mathbf{x}$ 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

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

- 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*.

Training GANs

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

---

for number of training iterations **do**

  for k steps **do**

  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.

  • Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.

  • Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right].$$

  end for

  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.

  • Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

Goodfellow et al. 2014

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)?

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

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* will output ~0.

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

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* will output ~0.

$$\begin{aligned}\nabla_{\theta_G} \log(1 - D(G(\mathbf{z}))) &= -\frac{1}{1 - D(G(\mathbf{z}))}\frac{\partial D}{\partial G}\frac{\partial G}{\partial \theta_G} \\ &= -\frac{1}{1}\sigma'(v)\frac{\partial v}{\partial G}\frac{\partial G}{\partial \theta_G}\end{aligned}$$

Here we assume *D* uses a logistic sigmoid *σ* as its output layer, whose input is *v*.

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* will output ~0.

$$\begin{aligned}
\nabla_{\theta_G} \log(1 - D(G(\mathbf{z}))) &= -\frac{1}{1 - D(G(\mathbf{z}))} \frac{\partial D}{\partial G} \frac{\partial G}{\partial \theta_G} \\
&= -\frac{1}{1} \sigma'(v) \frac{\partial v}{\partial G} \frac{\partial G}{\partial \theta_G} \\
&\approx -1 \times 0 \times \frac{\partial v}{\partial G} \frac{\partial G}{\partial \theta_G}
\end{aligned}$$

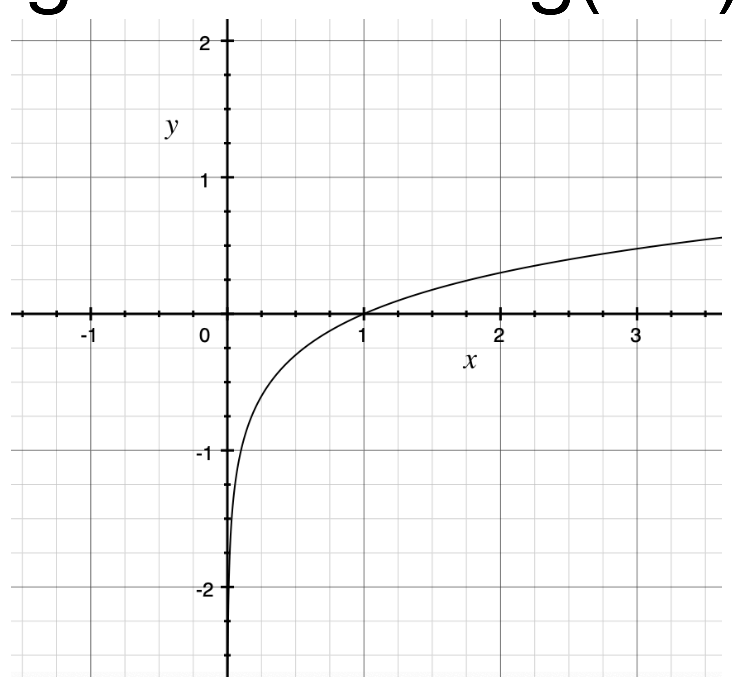
Closer look at *f*acc

- 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.

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

New loss term

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

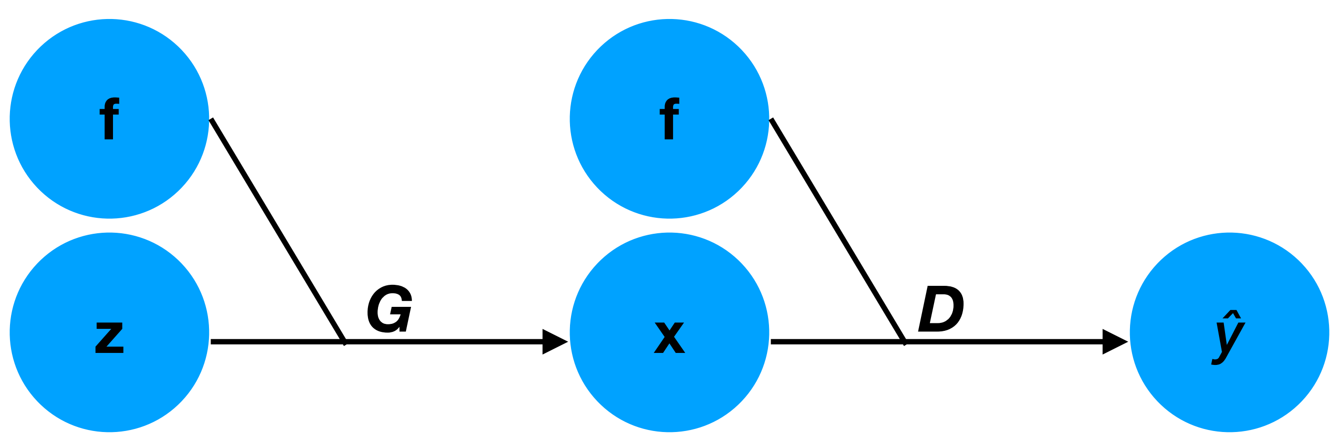


GANs

- Show Goodfellow et al. 2014 paper.
- GANs represent the state-of-the-art (as of 2021) for generating realistic data.
- GANs have also inspired many other adversarial training methods.

Conditional GANs

- One example is **conditional GANs:**
  - *G* also accepts a parameter 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.



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.:

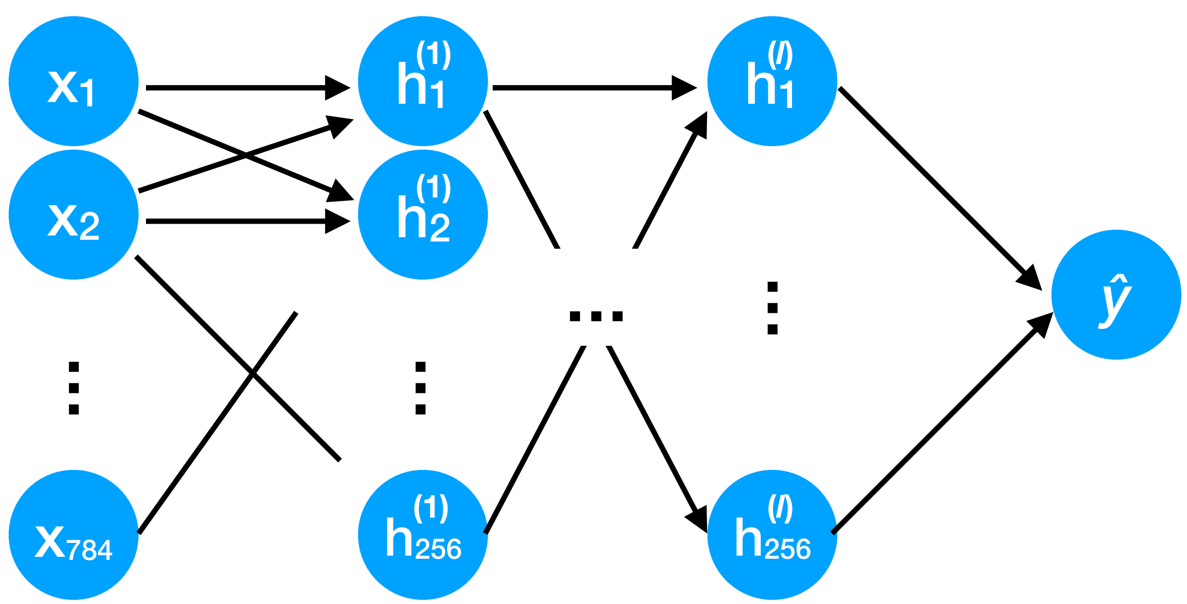
https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/

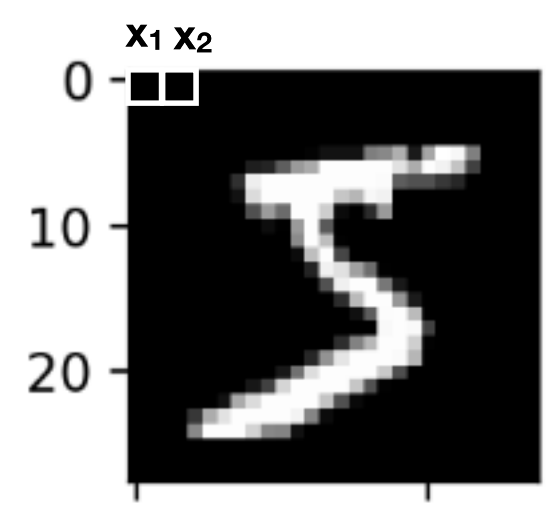
Difficulty in training

- GANs are renowned for being difficult to train:

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

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



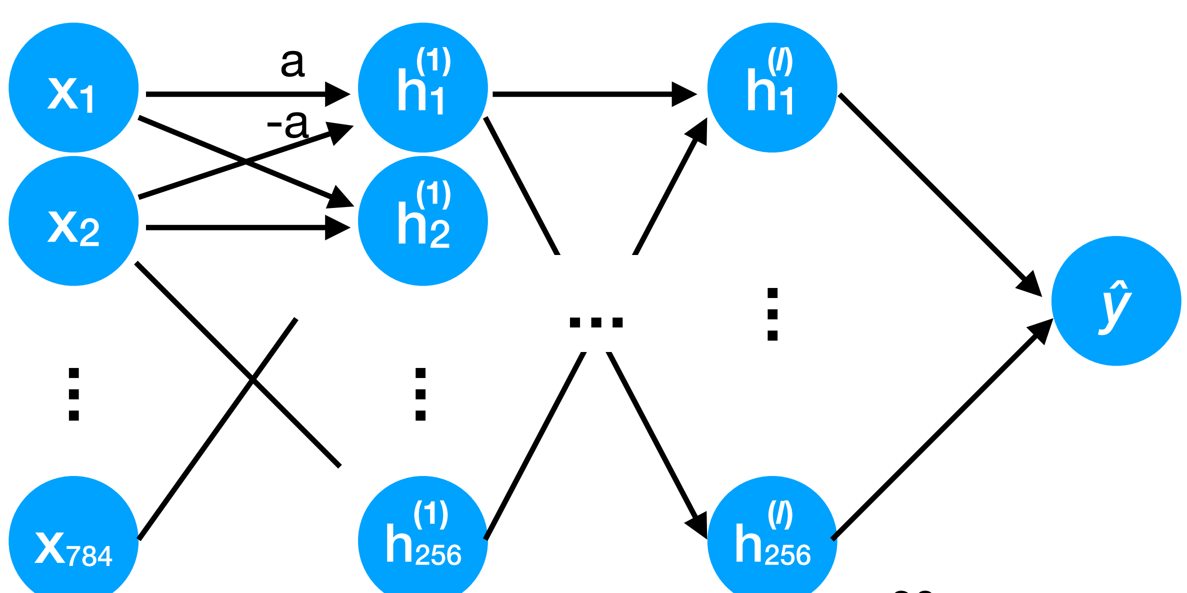


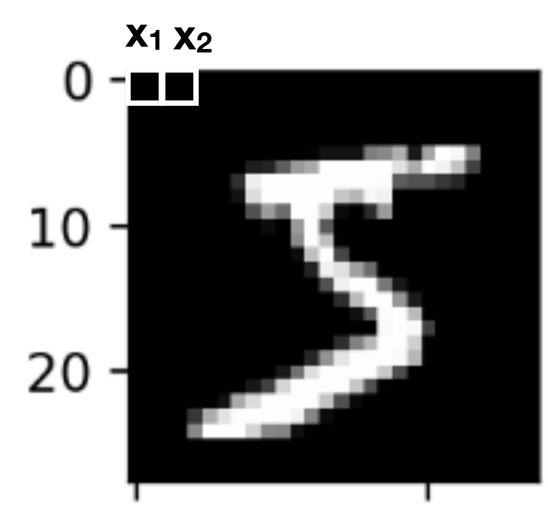
https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/

Difficulty in training

- GANs are renowned for being difficult to train:

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





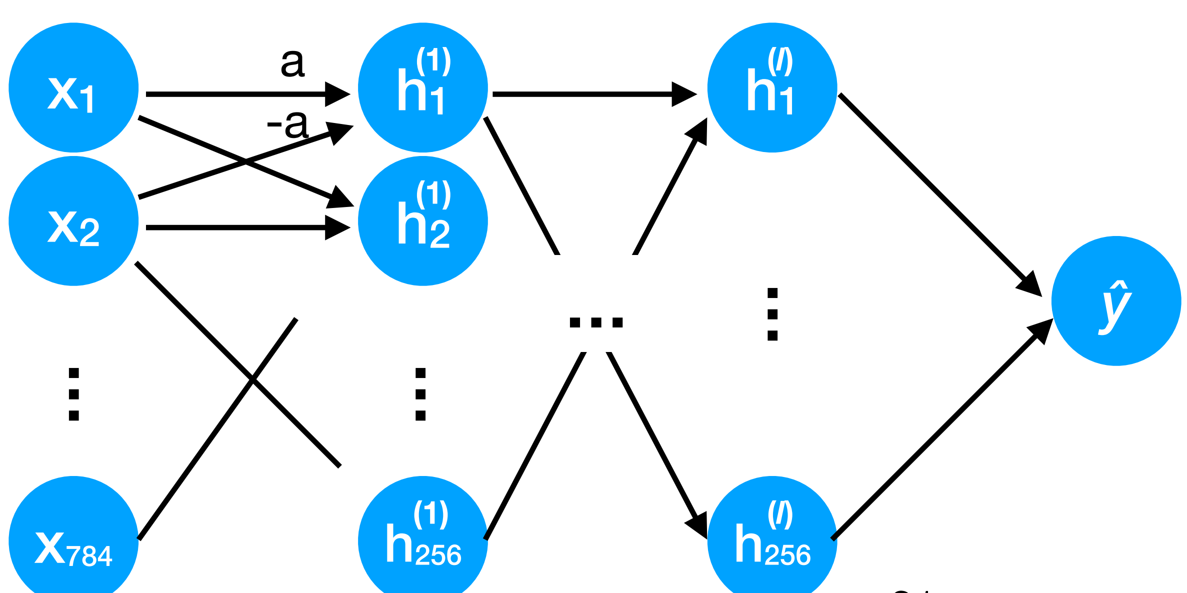
https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/

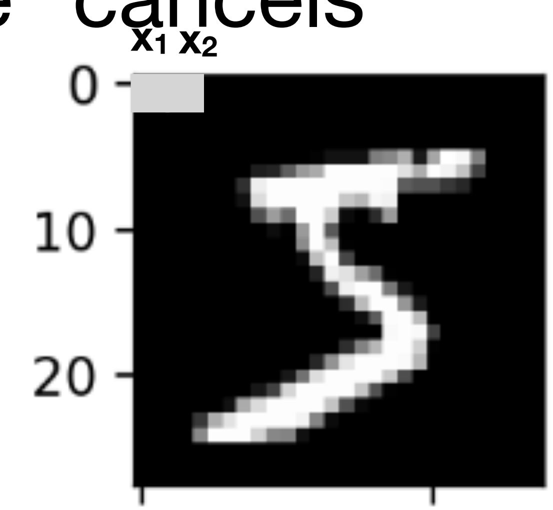
Difficulty in training

- GANs are renowned for being difficult to train:

  3. **Neuron co-adaptation** — training gets stuck because multiple 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.:





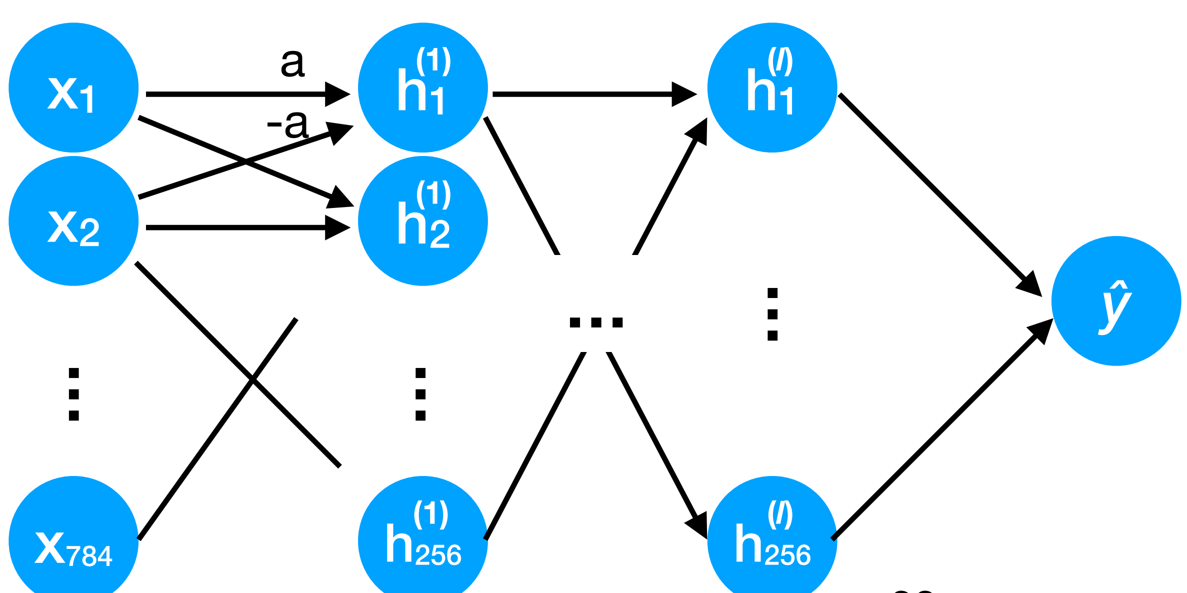
https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/

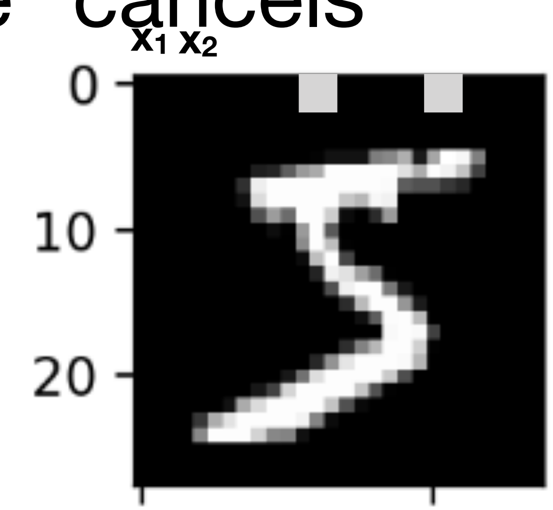
Difficulty in training

- GANs are renowned for being difficult to train:

  3. **Neuron co-adaptation** — training gets stuck because multiple 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.:





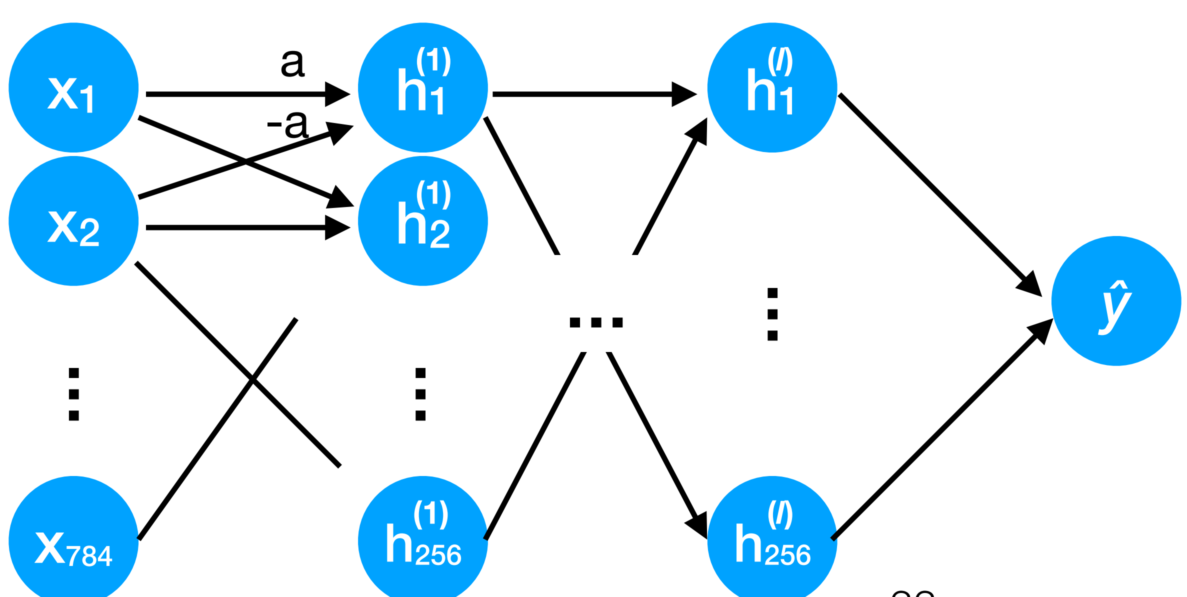
https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/

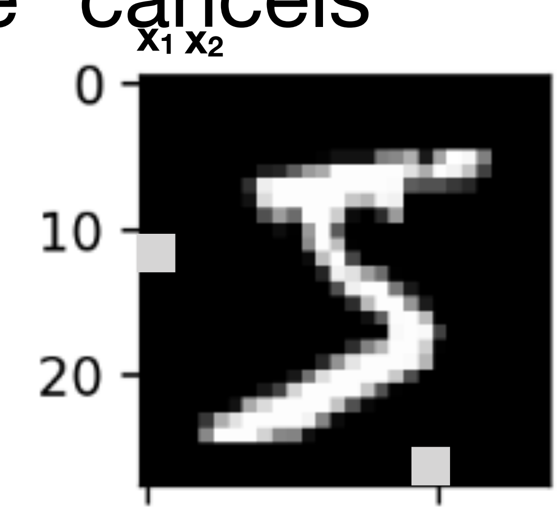
Difficulty in training

- GANs are renowned for being difficult to train:

  3. **Neuron co-adaptation** — training gets stuck because multiple 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.:





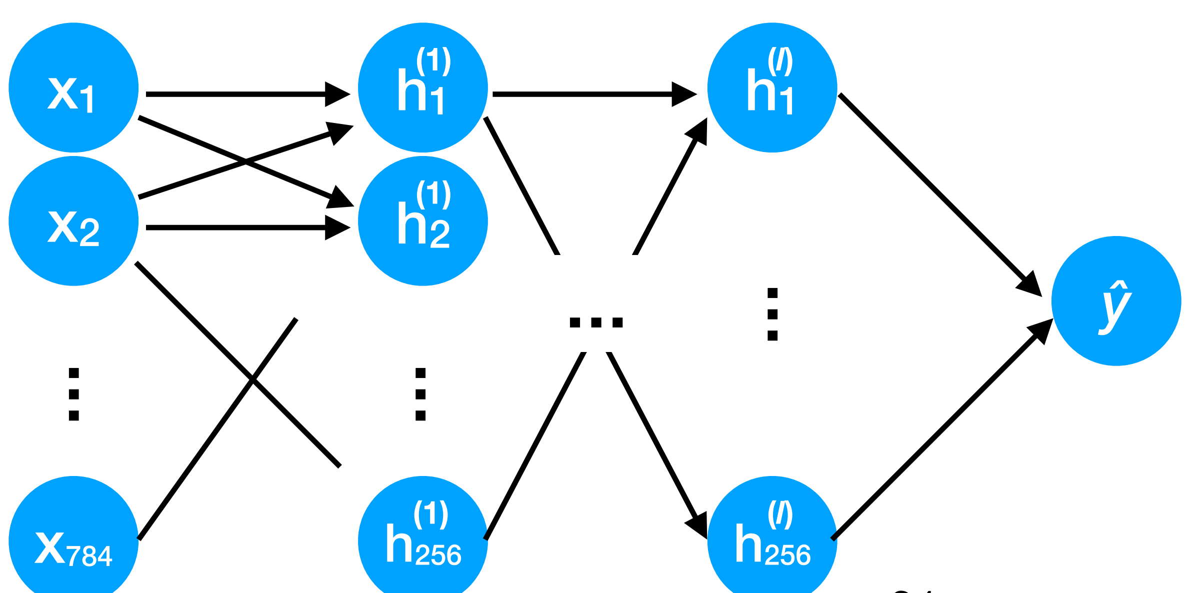
https://machinelearningmastery.com/practical-guide-to-gan-failure-modes/

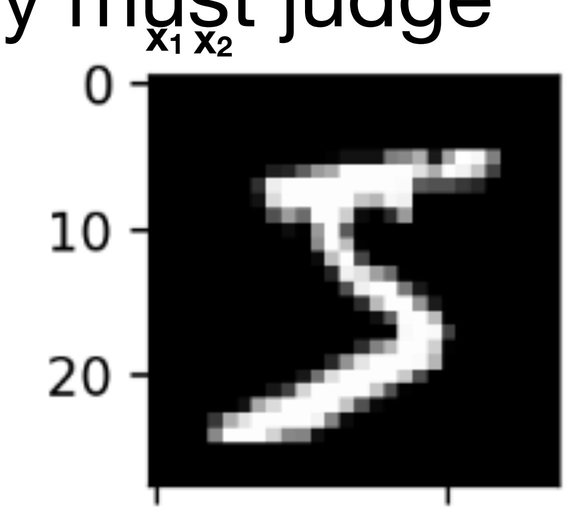
Difficulty in training

- GANs are renowned for being difficult to train:

  3. **Neuron co-adaptation** — training gets stuck because multiple 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/

Adversarial training examples

Adversarial training examples

- One of the weaknesses of DL is that we often do not know how the models work.

- This means that models can sometimes be systematically exploited to give nonsensical outputs.

- In high-stakes scenarios such as autonomous vehicles, the results can be disastrous (e.g., misread a speed limit sign of “25km/h” for “95km/h”).

Adversarial training examples

- One way to systematically confuse a trained NN is to “adjust” an input **x** so as to *maximally change* the output $\hat{y}$.

- In other words, for any loss function *f* (e.g., cross-entropy), compute the gradient:

$$\nabla_{\mathbf{x}} f(\mathbf{x}; \mathbf{w})$$

  where **w** are the NN's weights.

- We then construct an **adversarial example** (that differs by at most η from **x**) as:

$$\mathbf{x}' = \mathbf{x} + \eta \text{sign}(\nabla_{\mathbf{x}} f(\mathbf{x}; \mathbf{w}))$$

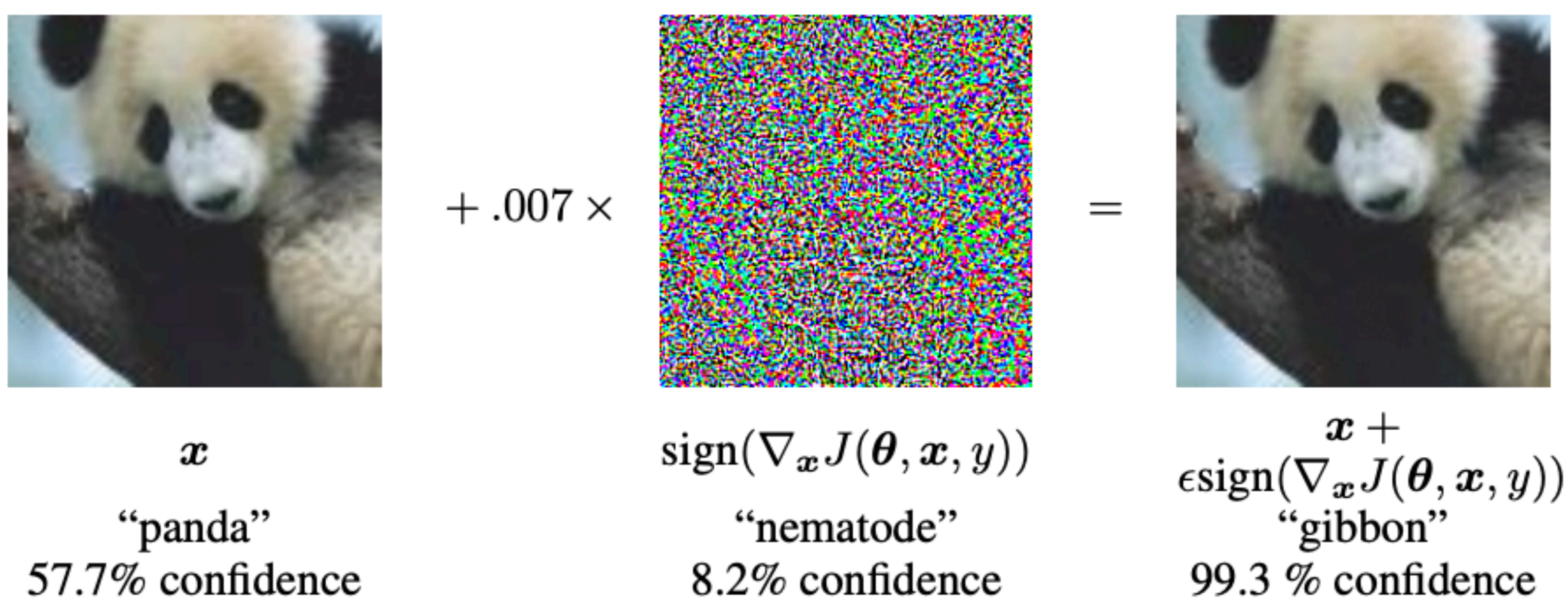
Adversarial training examples

- Note that this is *different* from what we usually do during training, i.e., conduct SGD on $\mathbf{w}$ using the gradient:

$$\nabla_{\mathbf{w}} f(\mathbf{x}; \mathbf{w})$$

Adversarial training examples

- This technique is responsible for the famous panda -> gibbon mistake (Goodfellow et al. 2014):



Adversarial training examples

- Since this discovery of this attack, many DL researchers have investigated how to prevent it.
- One simple technique that can help is to *train* explicitly on adversarial examples.
- This is easy, since we can create adversarial examples at will, and can improve robustness to outliers at test time.