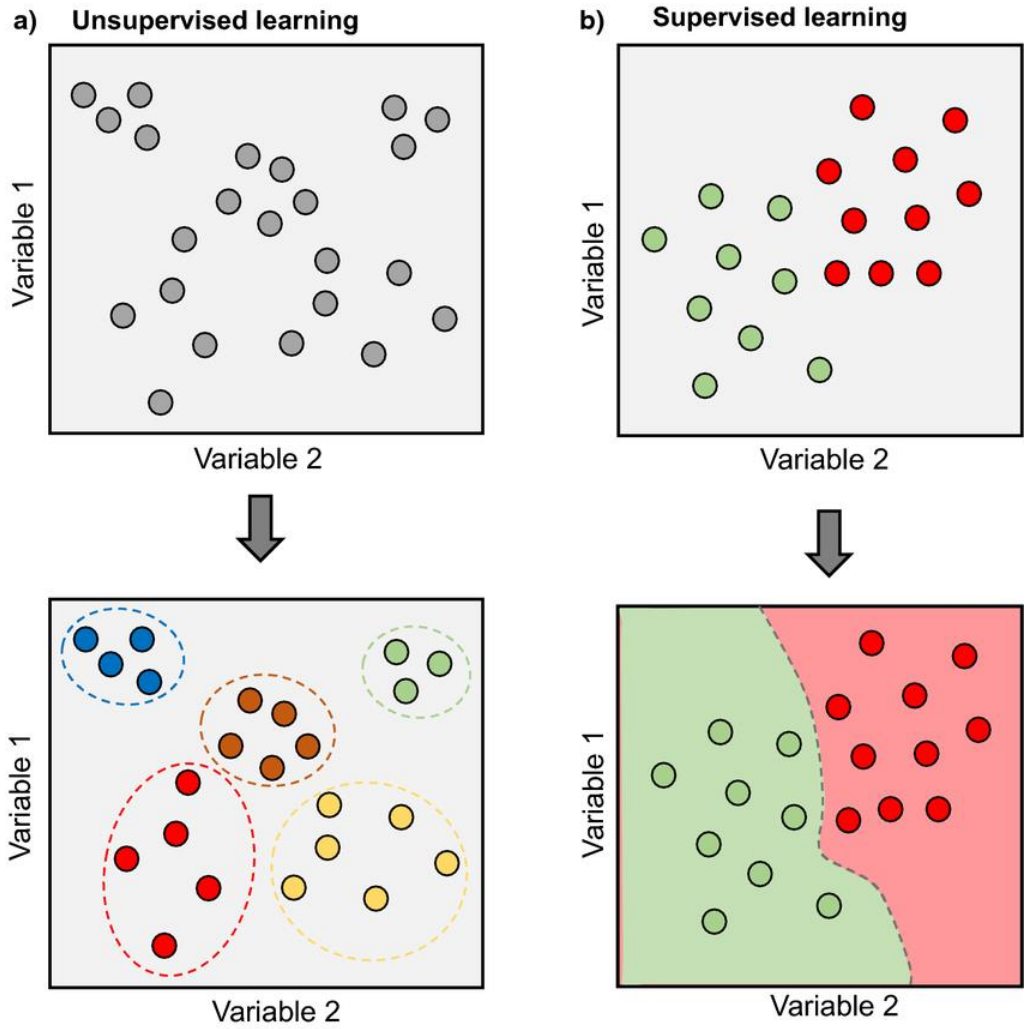
CSCE 633: Machine Learning

Lecture 37: Unsupervised Learning with Neural Networks

Texas A&M University
Bobak Mortazavi

Review

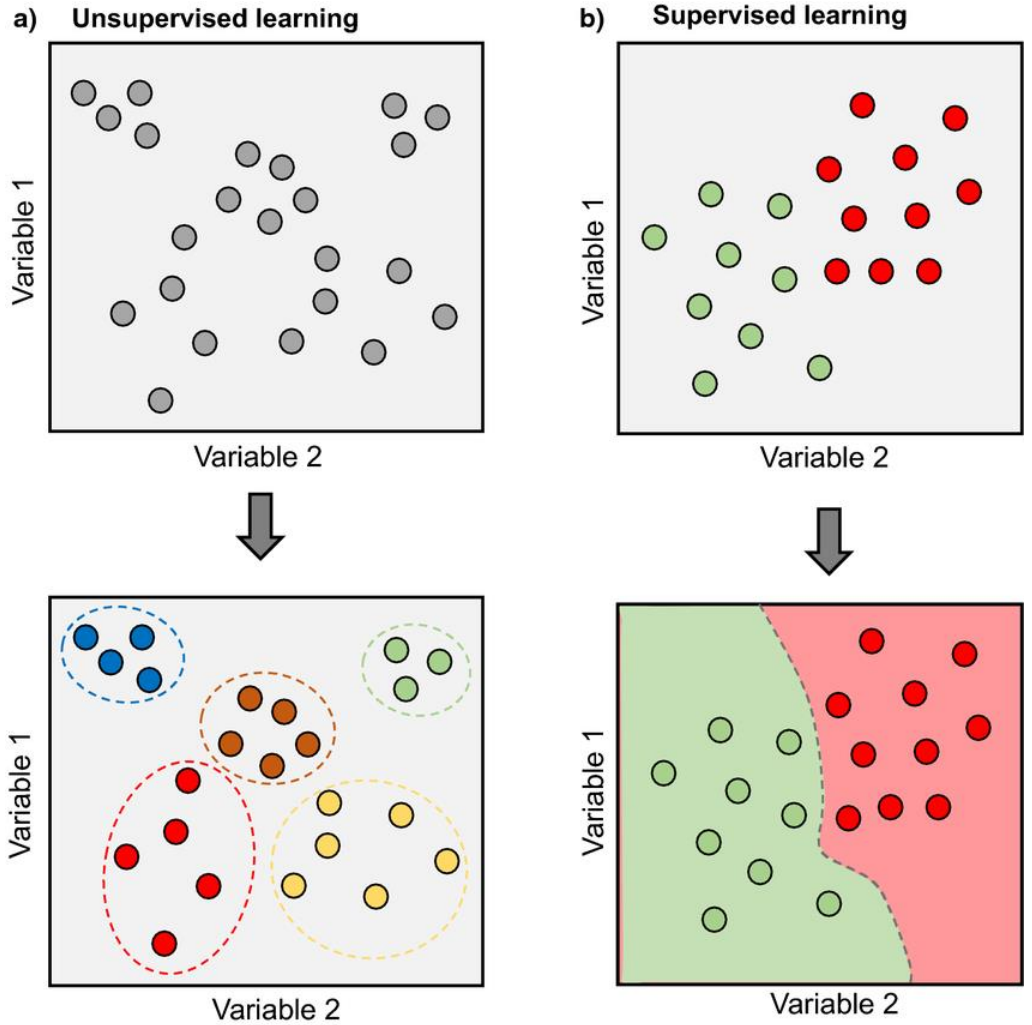
- Supervised vs unsupervised modeling



Morimoto, J., & Ponton, F. (2021). Virtual reality in biology: could we become virtual naturalists?. *Evolution: Education and Outreach*, *14*(1), 7.

Review

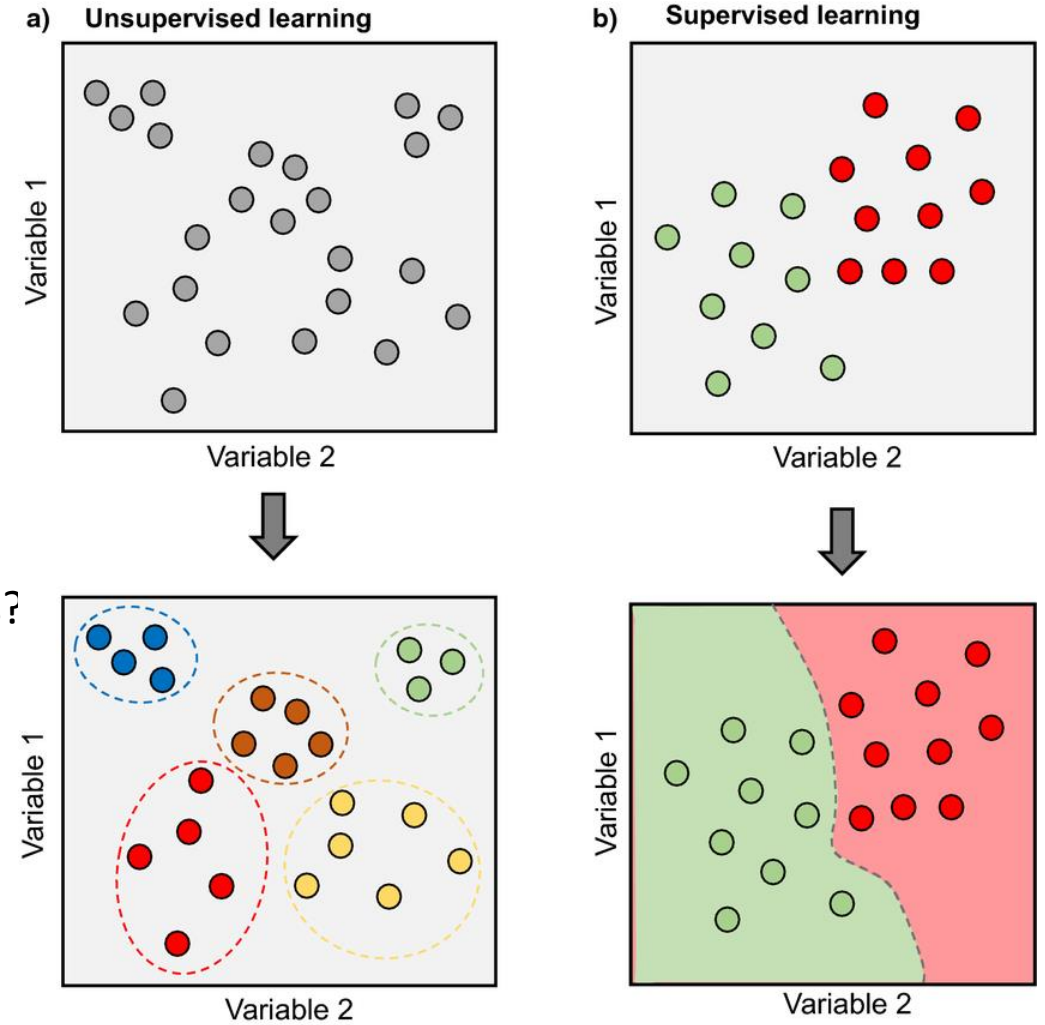
- Supervised methods:
  - Linear and logistic regression
  - Tree-based modeling
  - SVM
  - Supervised deep learning
- Unsupervised methods:
  - Clustering
  - Generative models



Morimoto, J., & Ponton, F. (2021). Virtual reality in biology: could we become virtual naturalists?. *Evolution: Education and Outreach*, *14*(1), 7.

Review

- Supervised methods:
  - Linear and logistic regression
  - Tree-based modeling
  - SVM
  - Supervised deep learning
- Unsupervised methods:
  - Clustering
  - Generative models
- What about semi-supervised and self-supervised methods?



Morimoto, J., & Ponton, F. (2021). Virtual reality in biology: could we become virtual naturalists?. *Evolution: Education and Outreach*, *14*(1), 7.

Unsupervised Learning with Deep Learning

- Discriminative vs Generative modeling – informal definition:
  - **Generative** models can generate new data instances.
  - **Discriminative** models discriminate between different kinds of data instances.

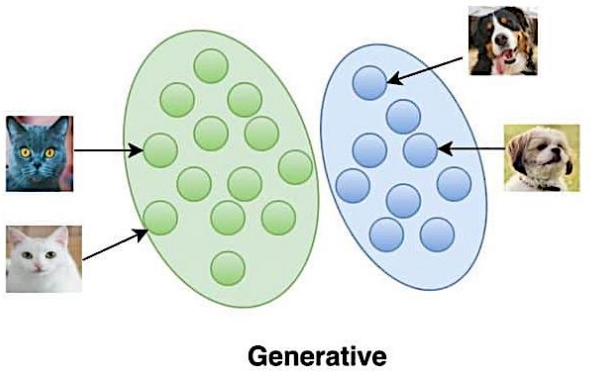
A generative model could generate new photos of animals that look like real animals, while a discriminative model could tell a dog from a cat.

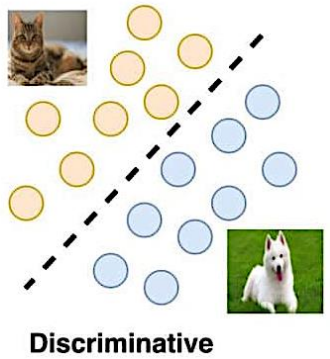
Discriminative vs Generative modeling – formal definition:

- **Generative** models capture the joint probability p(X, Y), or just p(X) if there are no labels.
- **Discriminative** models capture the conditional probability p(Y | X).

A generative model includes the distribution of the data itself, and tells you how likely a given example is.

A discriminative model ignores the question of whether a given instance is likely, and just tells you how likely a label is to apply to the instance.





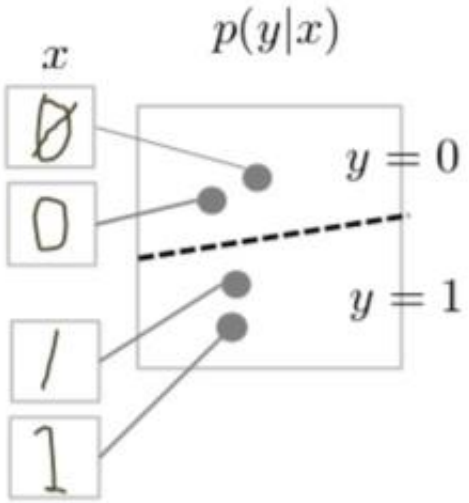
Source: https://developers.google.com/machine-learning/gan/generative
https://learnopencv.com/generative-and-discriminative-models/

Generative Models are Hard!

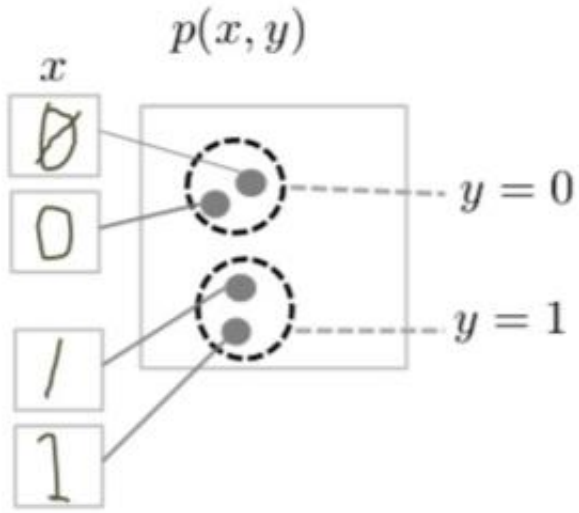
- Generative models tackle a more difficult task than analogous discriminative models. Generative models have to model *more*.
- A generative model for images might capture correlations like "things that look like boats are probably going to appear near things that look like water" and "eyes are unlikely to appear on foreheads." These are very complicated distributions.
- In contrast, a discriminative model might learn the difference between "sailboat" or "not sailboat" by just looking for a few tell-tale patterns. It could ignore many of the correlations that the generative model must get right.
- Discriminative models try to draw boundaries in the data space, while generative models try to model how data is placed throughout the space.

Generative Models are Hard!

- Discriminative Model



- Generative Model

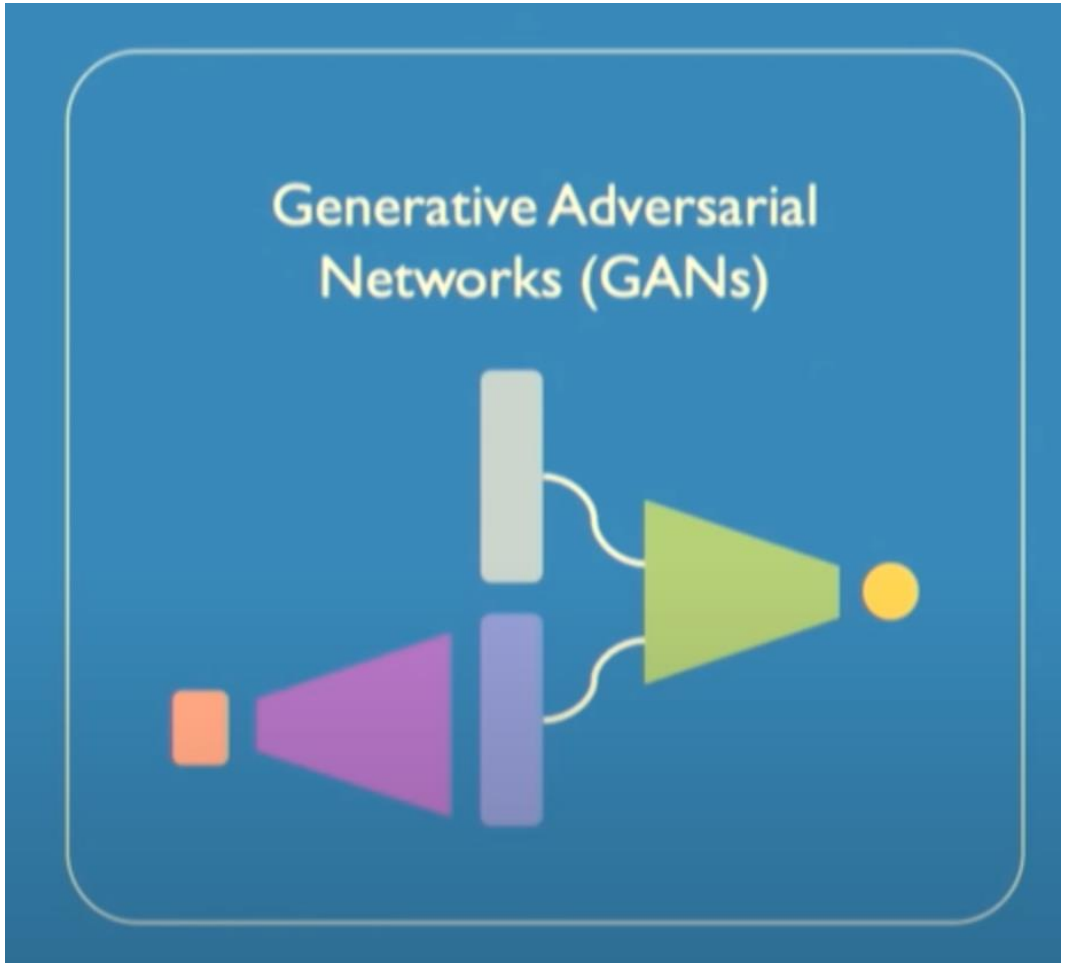


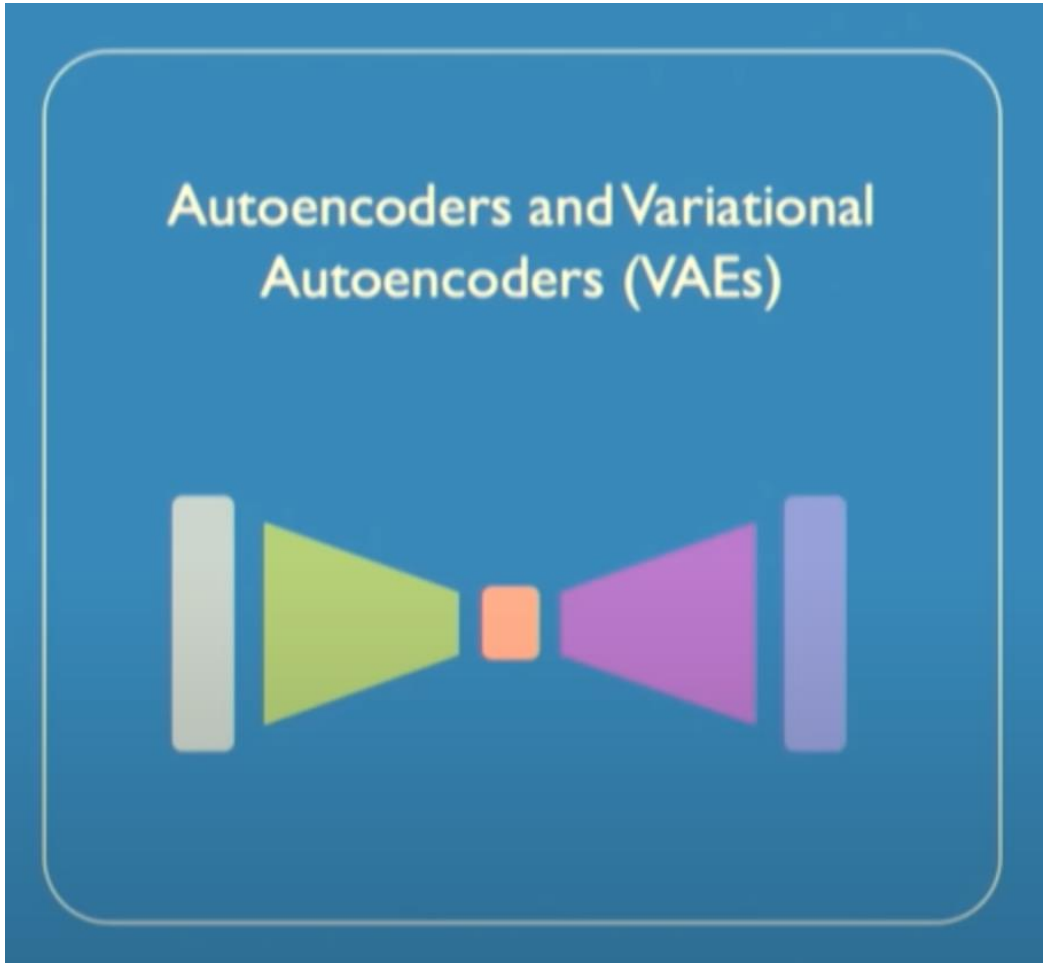
- The discriminative model tries to tell the difference between handwritten 0's and 1's by drawing a line in the data space. If it gets the line right, it can distinguish 0's from 1's without ever having to model exactly where the instances are placed in the data space on either side of the line.
- In contrast, the generative model tries to produce convincing 1's and 0's by generating digits that fall close to their real counterparts in the data space. It has to model the distribution throughout the data space.

Generative Models

- Generative Adversarial Network (GAN)
- Autoencoders and Variational Autoencoders (VAE)
- Masked Autoencoders
- Autoregressive Models
- Diffusion-based Deep Generative Models

Latent Variable Generative Models





Generative Adversarial Networks

Generative Adversarial Nets

Ian J. Goodfellow, Jean Pouget-Abadie,* Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair,† Aaron Courville, Yoshua Bengio‡
Département d'informatique et de recherche opérationnelle
Université de Montréal
Montréal, QC H3C 3J7

Abstract

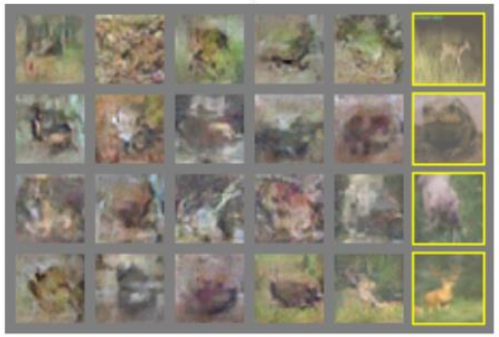
We propose a new framework for estimating generative models via an adversarial process, in which we simultaneously train two models: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G. The training procedure for G is to maximize the probability of D making a mistake. This framework corresponds to a minimax two-player game. In the space of arbitrary functions G and D, a unique solution exists, with G recovering the training data distribution and D equal to $\frac{1}{2}$ everywhere. In the case where G and D are defined by multilayer perceptrons, the entire system can be trained with backpropagation. There is no need for any Markov chains or unrolled approximate inference networks during either training or generation of samples. Experiments demonstrate the potential of the framework through qualitative and quantitative evaluation of the generated samples.

TITLE	CITED BY	YEAR
Generative adversarial networks I Goodfellow, J Pouget-Abadie, M Mirza, B Xu, D Warde-Farley, S Ozair, ... Advances in neural information processing systems 27	75745 *	2014

Visualization of samples from the model. The rightmost column shows the nearest training example of the neighboring sample

GAN trained on MNIST

GAN trained on CIFAR-10



Generative Adversarial Networks

A generative adversarial network (GAN) has two parts:

- The **generator** learns to generate plausible data. The generated instances become negative training examples for the discriminator.
- The **discriminator** learns to distinguish the generator's fake data from real data. The discriminator penalizes the generator for producing implausible results.

Generative Adversarial Networks

A generative adversarial network (GAN) has two parts:

- The **generator** learns to generate plausible data. The generated instances become negative training examples for the discriminator.
- The **discriminator** learns to distinguish the generator's fake data from real data. The discriminator penalizes the generator for producing implausible results.

When training begins, the generator produces obviously fake data, and the discriminator quickly learns to tell that it's fake:

As training progresses, the generator gets closer to producing output that can fool the discriminator:

Generative Adversarial Networks

A generative adversarial network (GAN) has two parts:

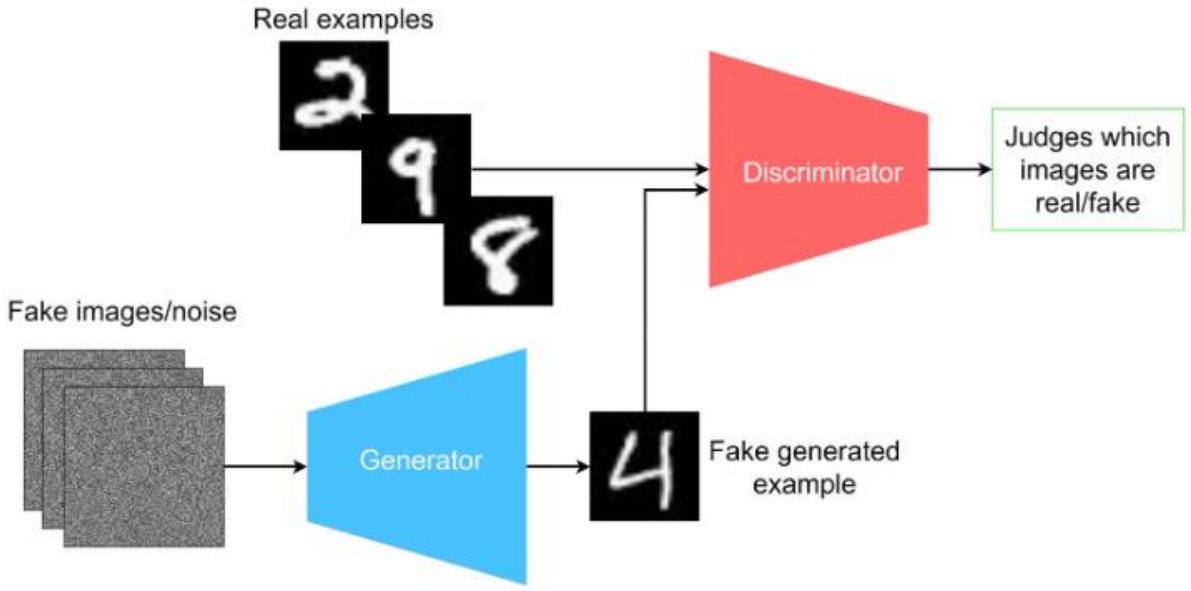
- The **generator** learns to generate plausible data. The generated instances become negative training examples for the discriminator.
- The **discriminator** learns to distinguish the generator's fake data from real data. The discriminator penalizes the generator for producing implausible results.

Finally, if generator training goes well, the discriminator gets worse at telling the difference between real and fake. It starts to classify fake data as real, and its accuracy decreases.

Generative Adversarial Networks

A generative adversarial network (GAN) has two parts:

- The **generator** learns to generate plausible data. The generated instances become negative training examples for the discriminator.
- The **discriminator** learns to distinguish the generator's fake data from real data. The discriminator penalizes the generator for producing implausible results.

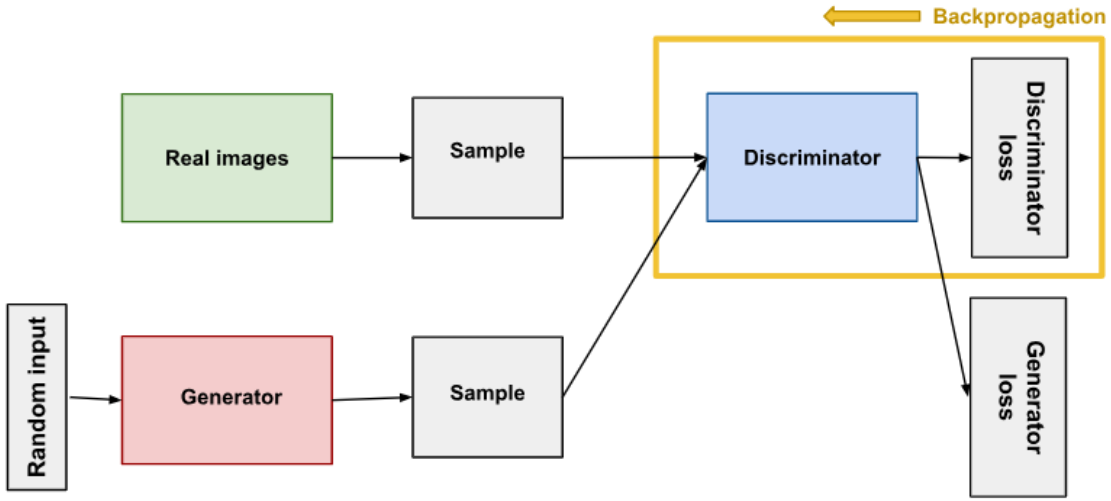


Discriminator

- The discriminator in a GAN is simply a classifier. It tries to distinguish real data from the data created by the generator. It could use any network architecture appropriate to the type of data it's classifying.
- The discriminator's training data comes from two sources:
- **Real data** instances, such as real pictures of people. The discriminator uses these instances as positive examples during training.
- **Fake data** instances created by the generator. The discriminator uses these instances as negative examples during training.

During discriminator training:

1.The discriminator classifies both real data and fake data from the generator.
2.The discriminator loss penalizes the discriminator for misclassifying a real instance as fake or a fake instance as real.
3.The discriminator updates its weights through backpropagation from the discriminator loss through the discriminator network.



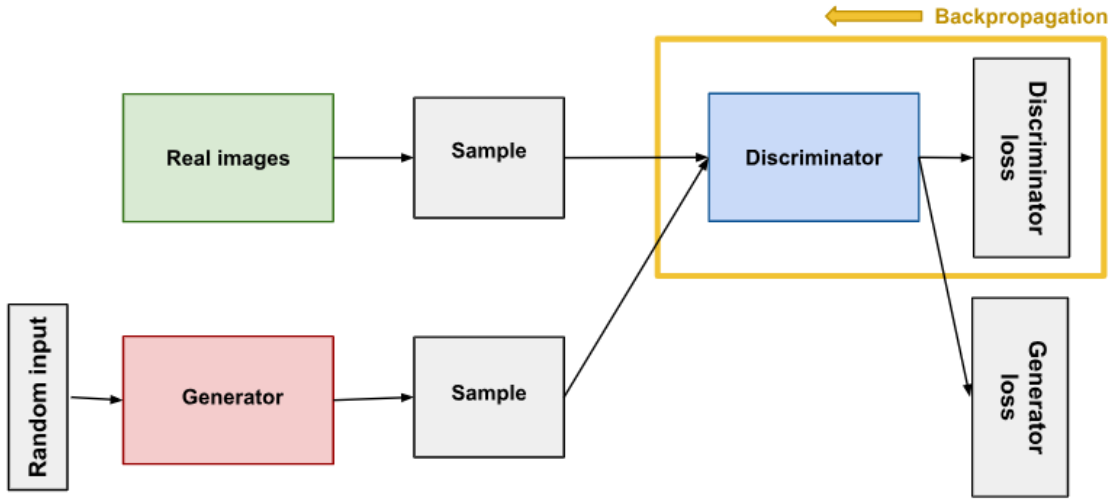
Discriminator Training

During discriminator training:

1.The discriminator classifies both real data and fake data from the generator.

2.The discriminator loss penalizes the discriminator for misclassifying a real instance as fake or a fake instance as real.

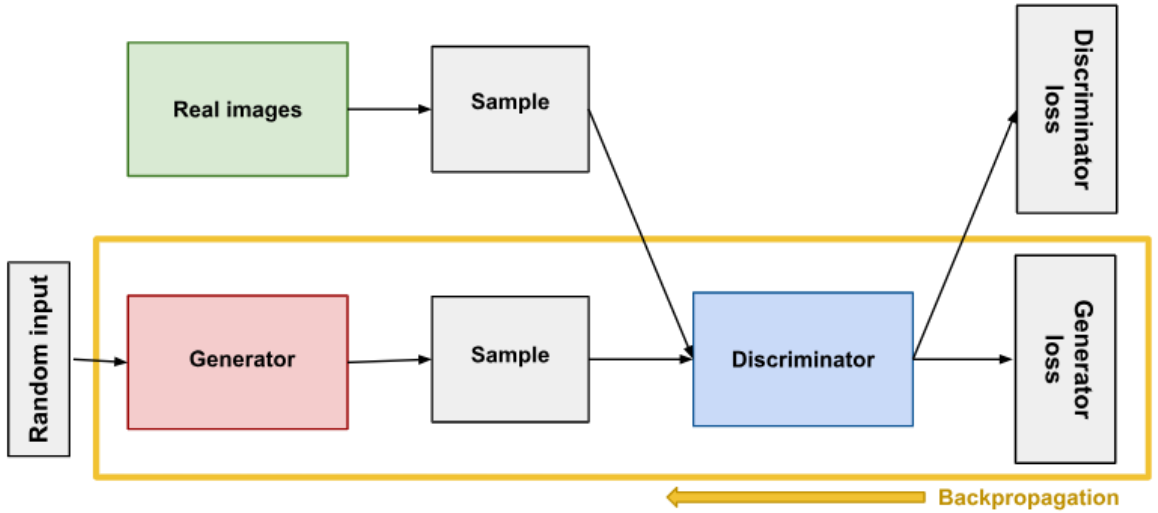
3.The discriminator updates its weights through backpropagation from the discriminator loss through the discriminator network.



$$\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

Generator

- The generator part of a GAN learns to create fake data by incorporating feedback from the discriminator. It learns to make the discriminator classify its output as real.
- Generator training requires tighter integration between the generator and the discriminator than discriminator training requires. The portion of the GAN that trains the generator includes:
  - random input
  - generator network, which transforms the random input into a data instance
  - discriminator network, which classifies the generated data
  - discriminator output
  - generator loss, which penalizes the generator for failing to fool the discriminator



Generator

- Generator training requires tighter integration between the generator and the discriminator than discriminator training requires. The portion of the GAN that trains the generator includes:
  - random input
  - generator network, which transforms the random input into a data instance
  - discriminator network, which classifies the generated data
  - discriminator output
  - generator loss, which penalizes the generator for failing to fool the discriminator
- **Why Random Input?**
  - Neural networks need some form of input. Normally we input data that we want to do something with, like an instance that we want to classify or make a prediction about. But what do we use as input for a network that outputs entirely new data instances?
  - In its most basic form, a GAN takes random noise as its input. The generator then transforms this noise into a meaningful output. By introducing noise, we can get the GAN to produce a wide variety of data, sampling from different places in the target distribution.
  - Experiments suggest that the distribution of the noise doesn't matter much, so we can choose something that's easy to sample from, like a uniform distribution. For convenience, the space from which the noise is sampled is usually of smaller dimension than the dimensionality of the output space.

Generator

- Generator training requires tighter integration between the generator and the discriminator than discriminator training requires. The portion of the GAN that trains the generator includes:
  - random input
  - generator network, which transforms the random input into a data instance
  - discriminator network, which classifies the generated data
  - discriminator output
  - generator loss, which penalizes the generator for failing to fool the discriminator
- **Why Random Input?**
  - Neural networks need some form of input. Normally we input data that we want to do something with, like an instance that we want to classify or make a prediction about. But what do we use as input for a network that outputs entirely new data instances?
  - In its most basic form, a GAN takes random noise as its input. The generator then transforms this noise into a meaningful output. By introducing noise, we can get the GAN to produce a wide variety of data, sampling from different places in the target distribution.
  - Experiments suggest that the distribution of the noise doesn't matter much, so we can choose something that's easy to sample from, like a uniform distribution. For convenience, the space from which the noise is sampled is usually of smaller dimension than the dimensionality of the output space.

Generator Training

- While the generator is trained, it samples random noise and produces an output from that noise. The output then goes through the discriminator and gets classified as either “Real” or “Fake” based on the ability of the discriminator to tell one from the other.
- The generator loss is then calculated from the discriminator’s classification – it gets rewarded if it successfully fools the discriminator, and gets penalized otherwise.

$$\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})}[\log D(\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{z}\sim p_{\boldsymbol{z}}(\boldsymbol{z})}[\log(1 - D(G(\boldsymbol{z})))]$$

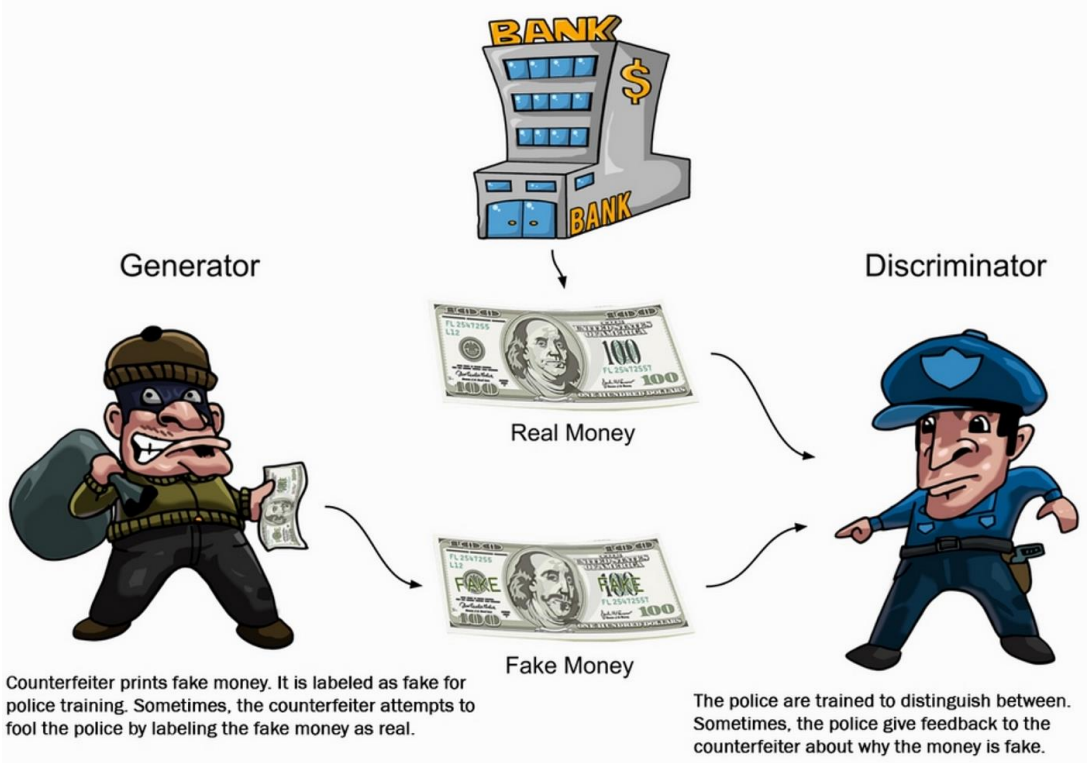
GAN Loss

- **Wait!** Same loss function for both the discriminator and generator?
- A GAN can have two loss functions: one for generator training and one for discriminator training. How can two loss functions work together to reflect a distance measure between probability distributions?

GAN Loss

- **Wait!** Same loss function for both the discriminator and generator?

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



GAN Loss: Minmax Loss

- In the paper that introduced GANs, the generator tries to minimize the following function while the discriminator tries to maximize it:

$$E_x[log(D(x))] + E_z[log(1 - D(G(z)))]$$

In this function:

- `D(x)` is the discriminator's estimate of the probability that real data instance x is real.
- E_x is the expected value over all real data instances.
- `G(z)` is the generator's output when given noise z.
- `D(G(z))` is the discriminator's estimate of the probability that a fake instance is real.
- E_z is the expected value over all random inputs to the generator (in effect, the expected value over all generated fake instances G(z)).
- The formula derives from the cross-entropy between the real and generated distributions.

The generator can't directly affect the `log(D(x))` term in the function, so, for the generator, minimizing the loss is equivalent to minimizing `log(1 - D(G(z)))`.

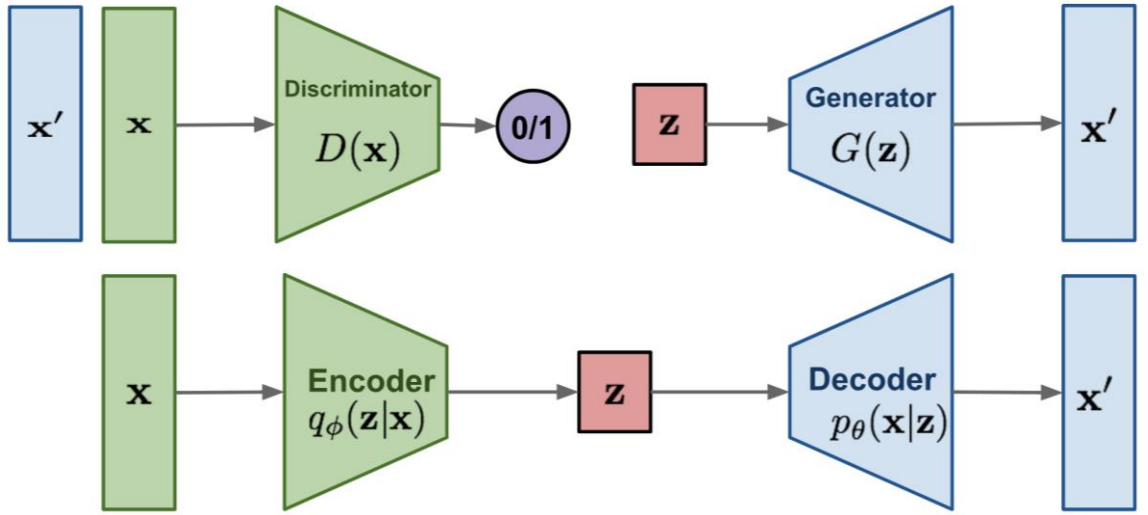
GAN: Common Problems

GANs have a number of common **failure modes**. All of these common problems are areas of active research.

- **Vanishing Gradients**
  - Research has suggested that if your discriminator is too good, then generator training can fail due to vanishing gradients. In effect, an optimal discriminator doesn't provide enough information for the generator to make progress.
- **Mode Collapse**
  - Usually you want your GAN to produce a wide variety of outputs. You want, for example, a different face for every random input to your face generator.
  - However, if a generator produces an especially plausible output, the generator may learn to produce *only* that output. In fact, the generator is always trying to find the one output that seems most plausible to the discriminator.
  - If the generator starts producing the same output (or a small set of outputs) over and over again, the discriminator's best strategy is to learn to always reject that output. But if the next generation of discriminator gets stuck in a local minimum and doesn't find the best strategy, then it's too easy for the next generator iteration to find the most plausible output for the current discriminator.
  - Each iteration of generator over-optimizes for a particular discriminator, and the discriminator never manages to learn its way out of the trap. As a result the generators rotate through a small set of output types. This form of GAN failure is called **mode collapse**.
- **Failure to Converge**
  - GANs frequently fail to converge, as discussed in the module on training.

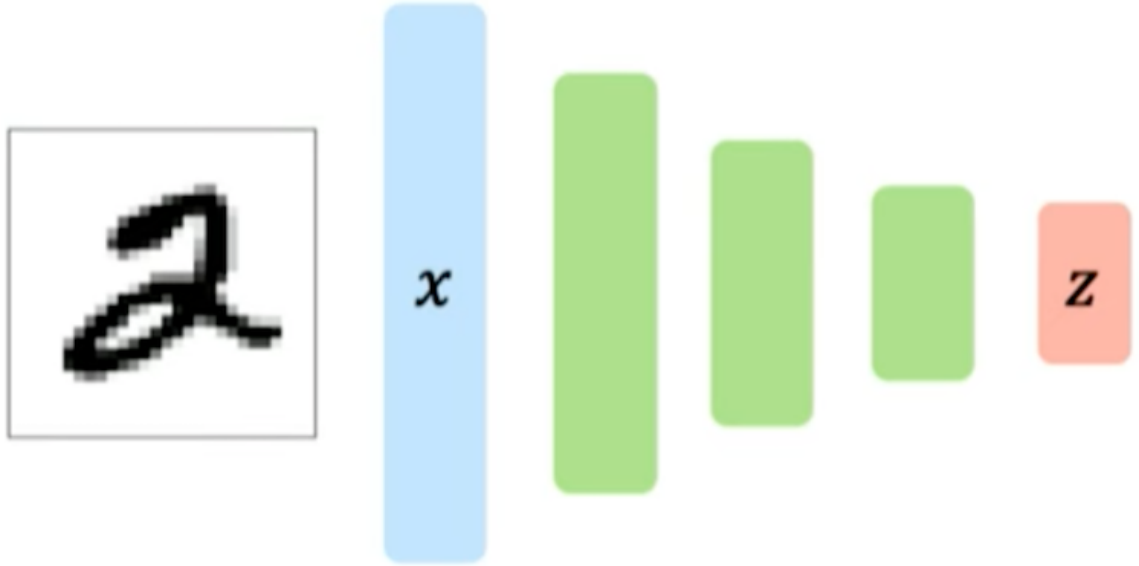
Autoencoder

- An autoencoder is a type of neural network architecture designed to efficiently **compress (encode)** input data down to its essential features, then **reconstruct (decode)** the original input from this compressed representation.
- Using unsupervised machine learning, autoencoders are trained to discover ***latent variables*** of the input data.
- Collectively, the latent variables of a given set of input data are referred to as *latent space*. During training, the autoencoder learns which latent variables can be used to most accurately reconstruct the original data: this latent space representation thus represents only the most essential information contained within the original input.
- Most types of autoencoders are used for artificial intelligence tasks related to feature extraction, like data compression, image denoising, anomaly detection and facial recognition.



Autoencoders: Background

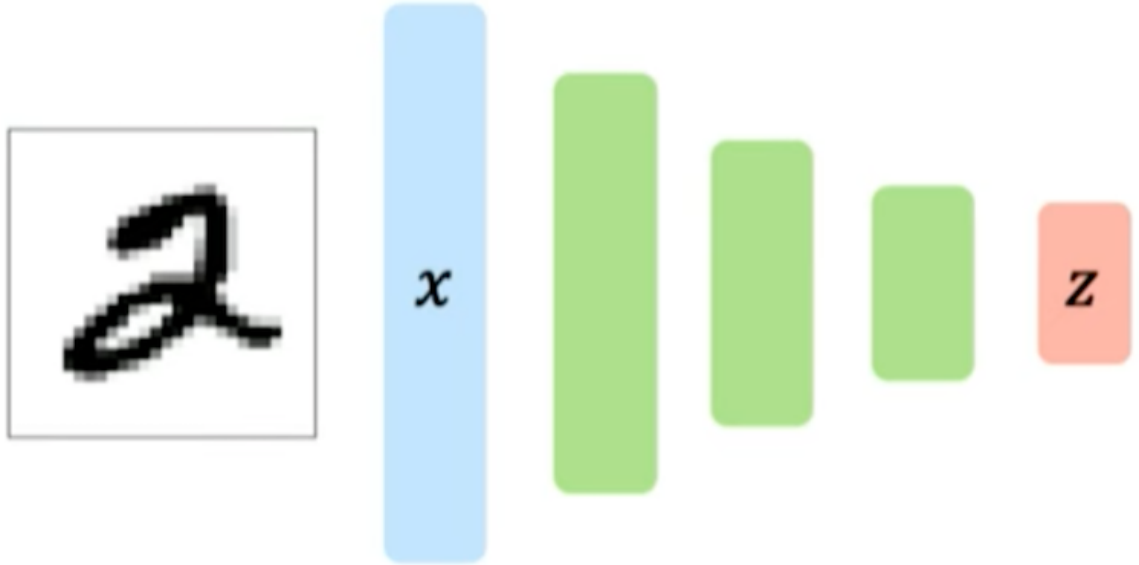
- Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data



- Encoder learns mapping from the data x to a low-dimensional latent space z

Autoencoders: Background

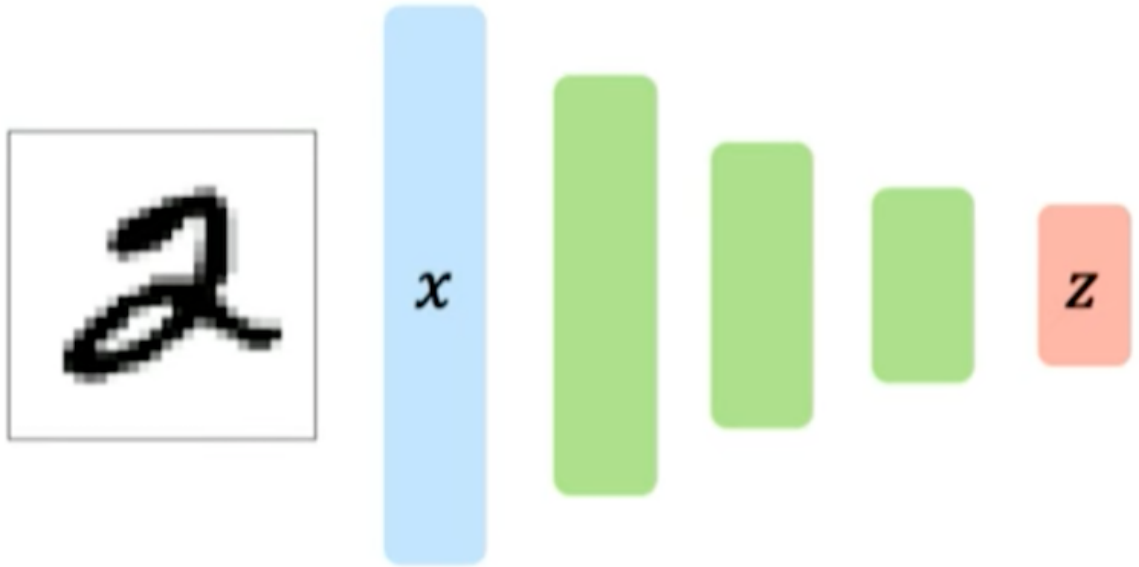
- Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data



- Encoder learns mapping from the data x to a low-dimensional latent space z
- **But why?** It's an unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

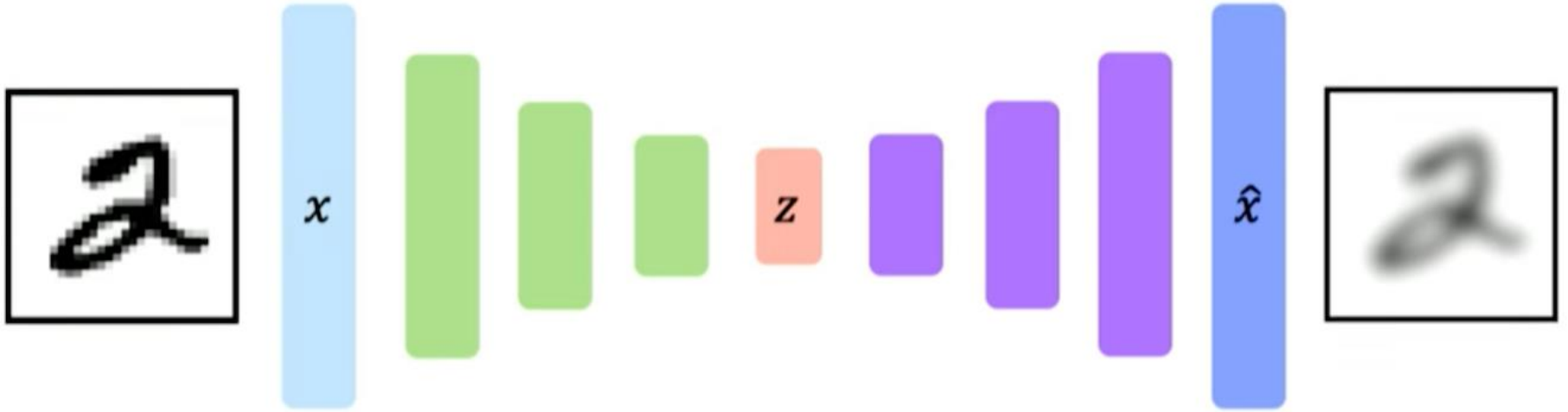
Autoencoders: Background

- How can we learn this latent space?
  - Train the model to use these features to reconstruct the original data
- Decoder learns mapping back from latent space z to a reconstructed observation $\hat{x}$



Autoencoders: Background

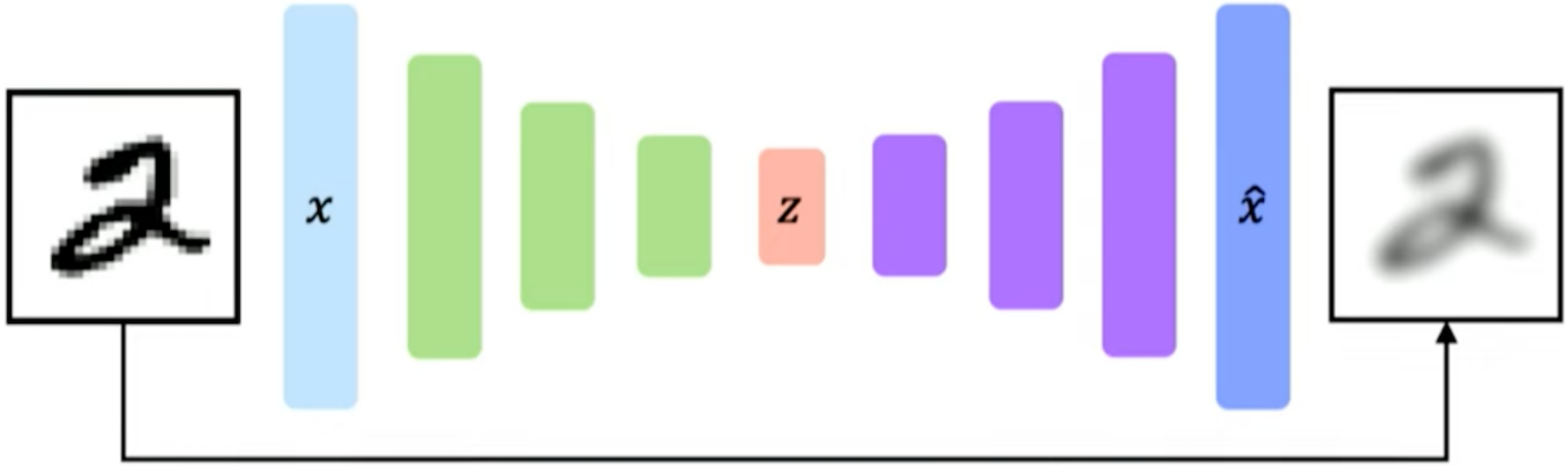
- How can we learn this latent space?
  - Train the model to use these features to reconstruct the original data
- Decoder learns mapping back from latent space z to a reconstructed observation $\hat{x}$



No labels! So how do we train this model?
What would be a good loss function here?

Autoencoders: Background

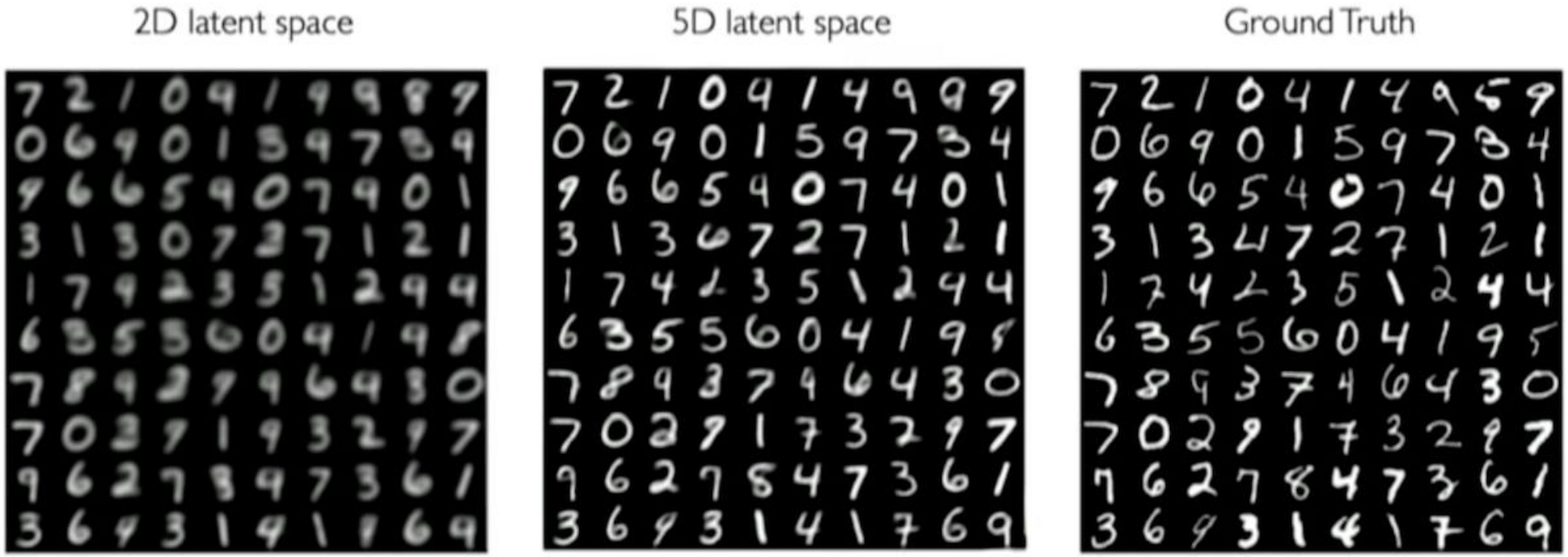
- How can we learn this latent space?
  - Train the model to use these features to reconstruct the original data
- Decoder learns mapping back from latent space z to a reconstructed observation $\hat{x}$



$$\mathcal{L}(x, \hat{x}) = \|x - \hat{x}\|^2$$

Dimensionality of latent space: Reconstruction Quality

- Autoencoding is a form of compression!
- Smaller latent space will force a larger training bottleneck

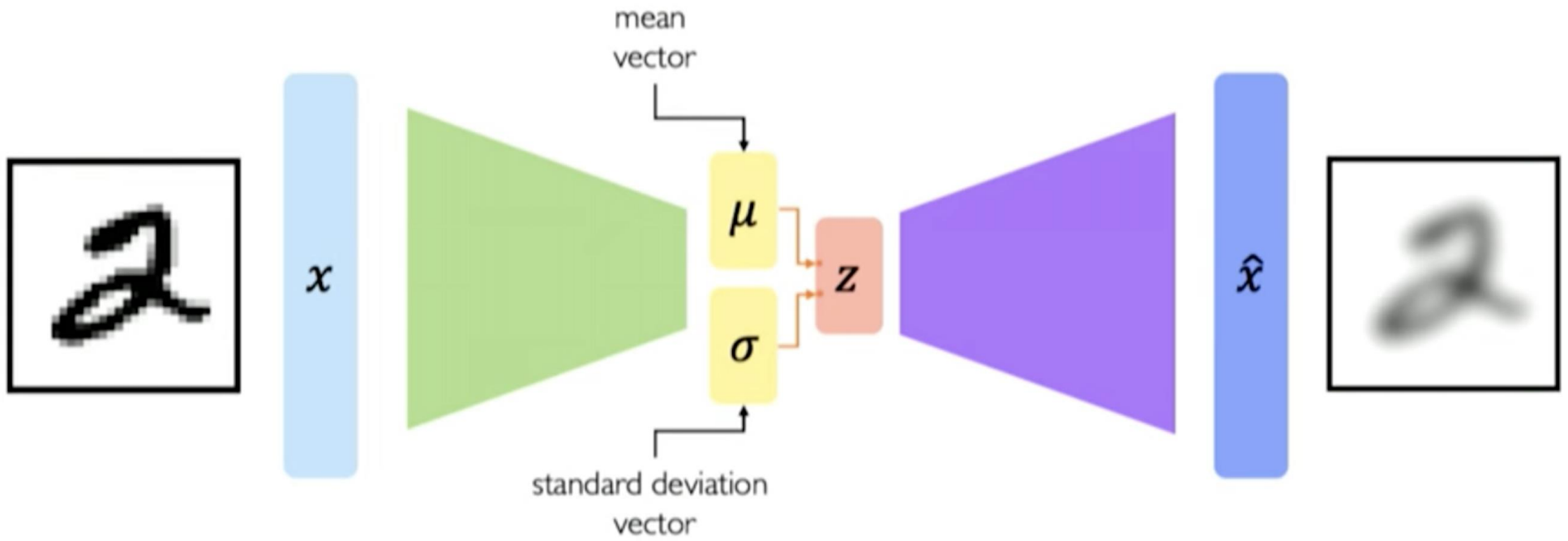


Autoencoders for representation learning

- **Bottleneck hidden layer** forces network to learn a compressed latent representation
- **Reconstruction loss** forces latent representation to capture or encode as much information about the data as possible
- **Autoencoding** = Automatically encoding the data

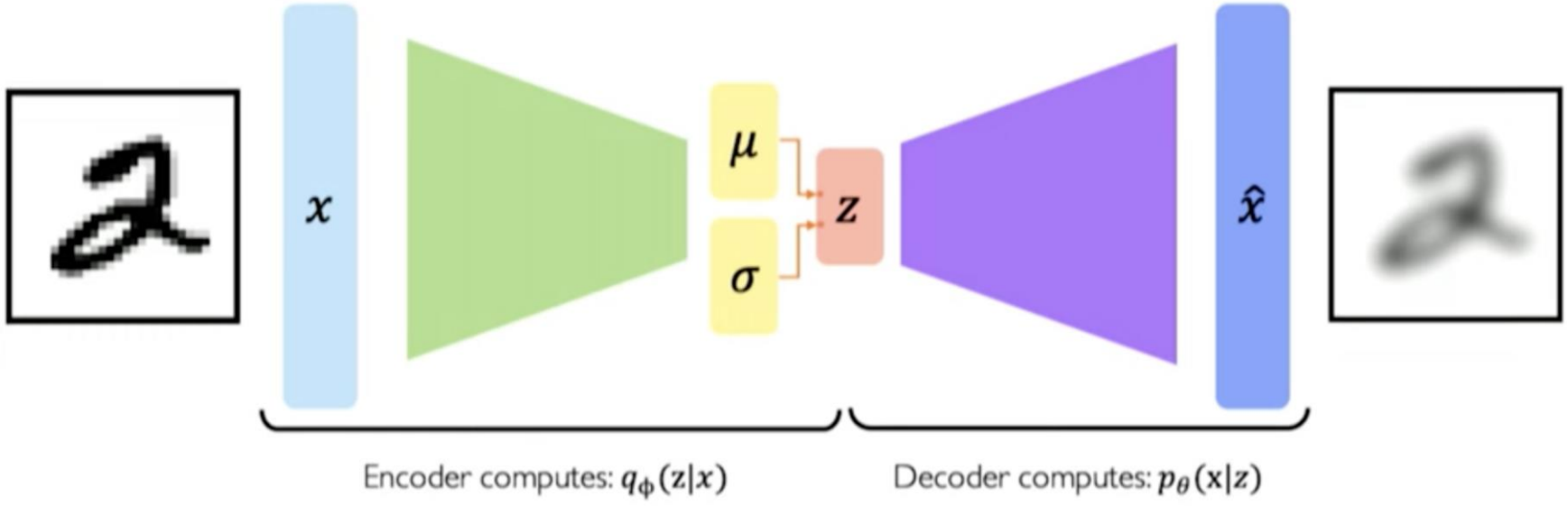
Variational Autoencoders

- Variational autoencoders are a probabilistic twist on autoencoders!
- Sample from mean and standard deviation to compute the latent space



Variational Autoencoders

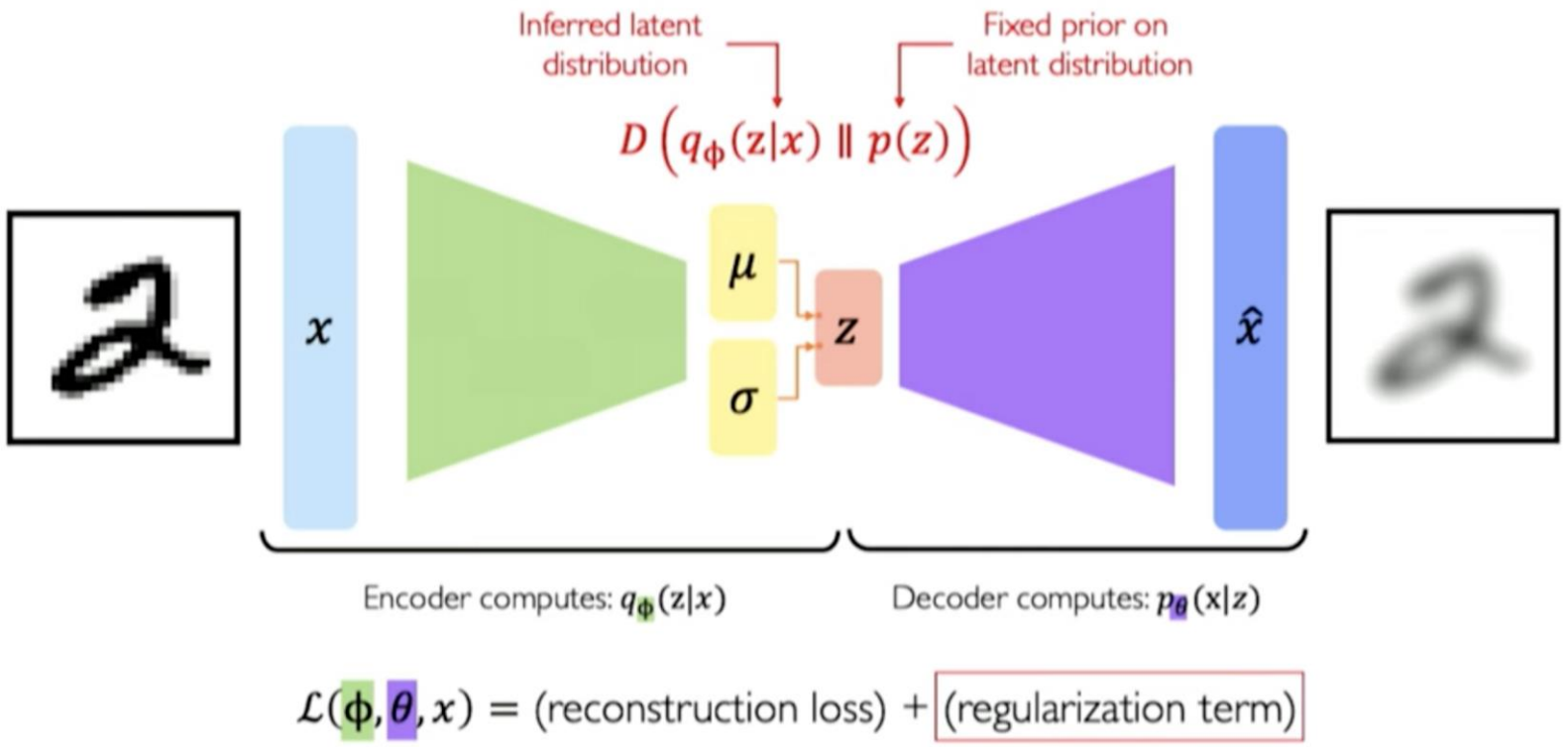
- Variational autoencoders are a probabilistic twist on autoencoders!
- Sample from mean and standard deviation to compute the latent space



$$\mathcal{L}(\phi, \theta, x) = \text{(reconstruction loss)} + \text{(regularization term)}$$

Variational Autoencoders

- Variational autoencoders are a probabilistic twist on autoencoders!
- Sample from mean and standard deviation to compute the latent space



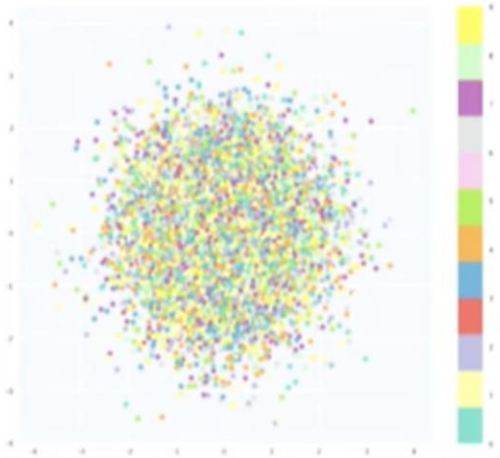
Variational Autoencoders: Priors on the latent distribuition

- Variational autoencoders are a probabilistic twist on autoencoders!
- Sample from mean and standard deviation to compute the latent space

$$D\left(q_\phi(\mathrm{z}|x) \parallel p(z)\right)$$

Inferred latent distribution

Fixed prior on latent distribution



Common choice of prior – Normal Gaussian:

$$p(z) = \mathcal{N}(\mu = 0, \sigma^2 = 1)$$

- Encourages encodings to distribute encodings evenly around the center of the latent space
- Penalize the network when it tries to "cheat" by clustering points in specific regions (i.e., by memorizing the data)

Variational Autoencoders: Priors on the latent distribuition

- Variational autoencoders are a probabilistic twist on autoencoders!
- Sample from mean and standard deviation to compute the latent space

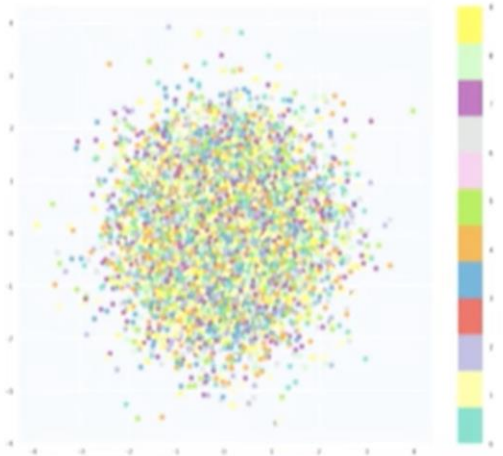
$$D\left(q_\phi(\mathrm{z}|x) \parallel p(z)\right)$$

Inferred latent distribution

Fixed prior on latent distribution

$$D\left(q_\phi(\mathrm{z}|x) \parallel p(z)\right)$$
$$= -\frac{1}{2}\sum_{j=0}^{k-1}(\sigma_j + \mu_j^2 - 1 - \log\sigma_j)$$

KL-divergence between the two distributions



Common choice of prior – Normal Gaussian:

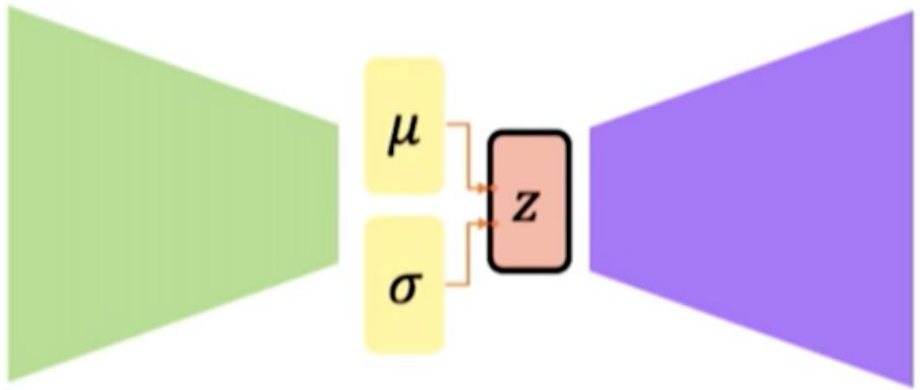
$$p(z) = \mathcal{N}(\mu = 0, \sigma^2 = 1)$$

- Encourages encodings to distribute encodings evenly around the center of the latent space
- Penalize the network when it tries to "cheat" by clustering points in specific regions (i.e., by memorizing the data)

Intuition on regularization and the Normal prior

1. Continuity: points that are close in latent space → similar content after decoding
2. Completeness: sampling from latent space → “meaningful” content after decoding

Reparametrizing the sampling layer



Key Idea:

~~$z \sim \mathcal{N}(\mu, \sigma^2)$~~

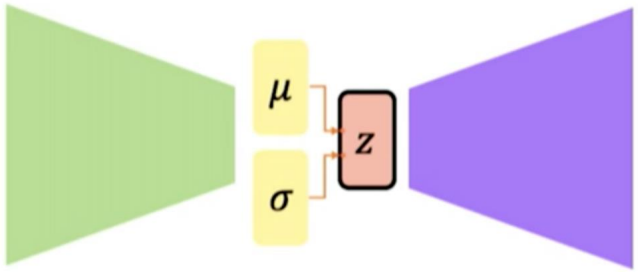
Consider the sampled latent vector z as a sum of

- a fixed μ vector,
- and fixed σ vector, scaled by random constants drawn from the prior distribution

$$\Rightarrow z = \mu + \sigma \odot \varepsilon$$

where $\varepsilon \sim \mathcal{N}(0,1)$

Reparametrizing the sampling layer



Key Idea:

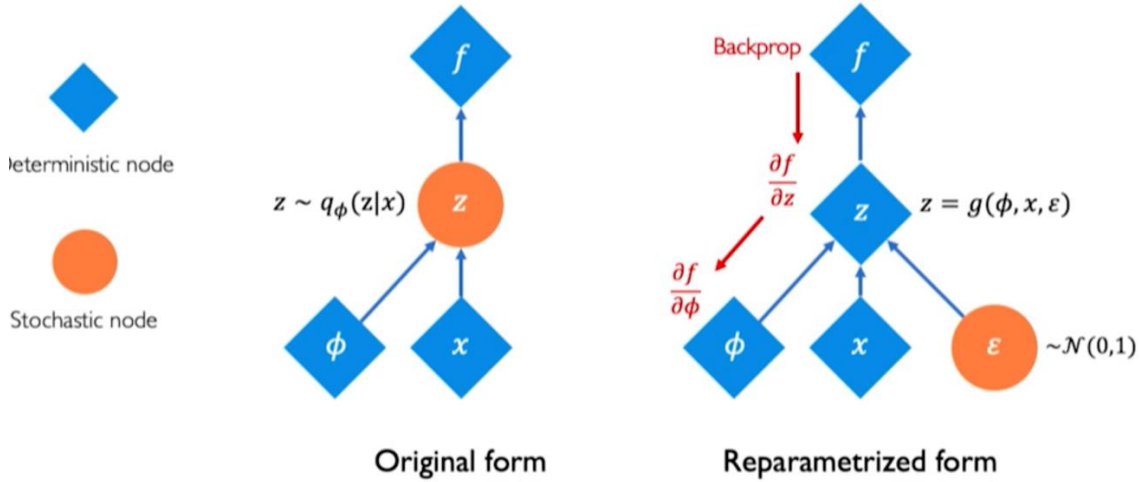
~~$z \sim \mathcal{N}(\mu, \sigma^2)$~~

Consider the sampled latent vector z as a sum of

- a fixed μ vector,
- and fixed σ vector, scaled by random constants drawn from the prior distribution

$$\Rightarrow z = \mu + \sigma \odot \varepsilon$$

where $\varepsilon \sim \mathcal{N}(0,1)$



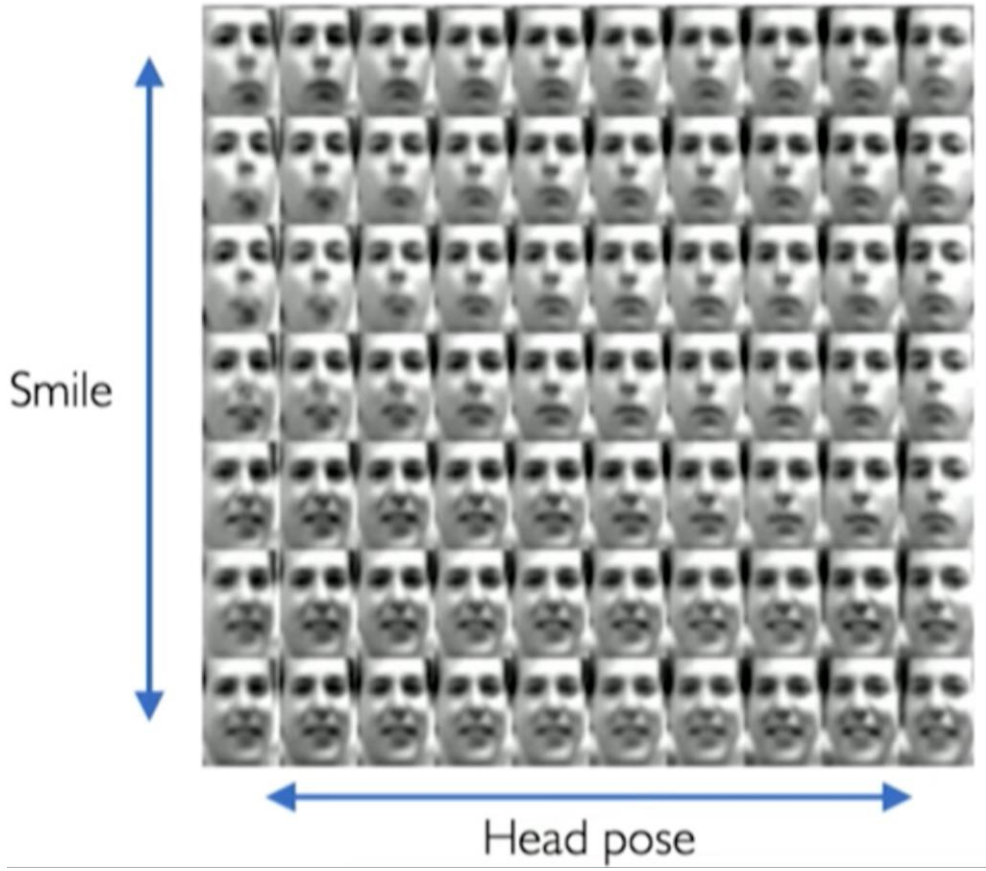
VAEs: Latent Perturbations

Slowly increase or decrease a **single latent variable**
Keep all other variables fixed

Head pose

Different dimensions of z encodes **different interpretable latent features**

VAEs: Latent Perturbations



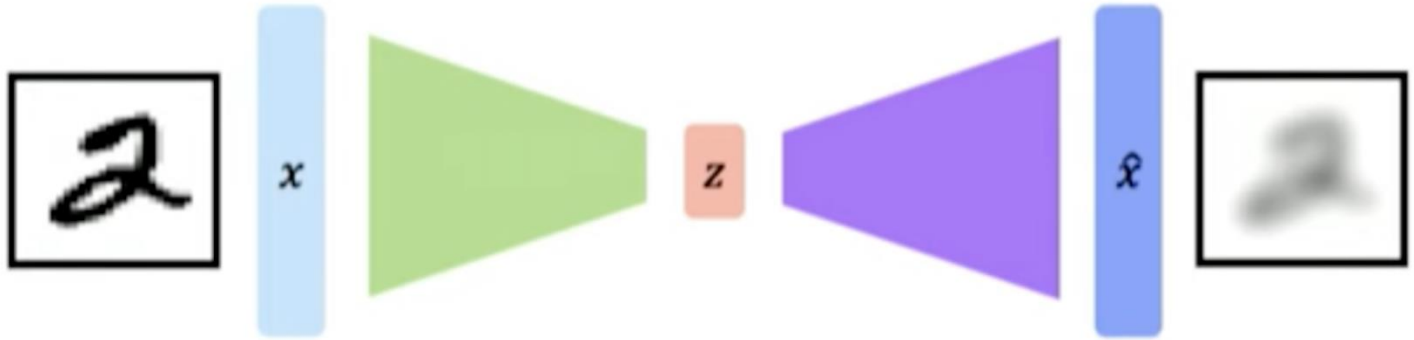
Ideally, we want latent variables that are uncorrelated with each other

Enforce diagonal prior on the latent variables to encourage independence

Disentanglement

VAE: Summary

1. Compress representation of world to something we use to learn
2. Reconstruction allows for unsupervised learning (no labels!)
3. Reparameterization trick to train end-to-end
4. Interpret hidden latent variables using perturbation
5. Generating new examples



Masked Autoencoders

Masked Autoencoders Are Scalable Vision Learners

Kaiming He*,† Xinlei Chen* Saining Xie Yanghao Li Piotr Dollár Ross Girshick

*equal technical contribution †project lead

Facebook AI Research (FAIR)

Abstract

This paper shows that masked autoencoders (MAE) are scalable self-supervised learners for computer vision. Our MAE approach is simple: we mask random patches of the input image and reconstruct the missing pixels. It is based on two core designs. First, we develop an asymmetric *encoder-decoder architecture, with an encoder that operates only on the visible subset of patches (without mask tokens), along with a lightweight decoder that reconstructs the original image from the latent representation and mask tokens. Second, we find that masking a high proportion of the input image,* e.g., *75%, yields a nontrivial and meaningful self-supervisory task. Coupling these two designs enables us to train large models efficiently and effectively: we accelerate training (by 3× or more) and improve accuracy. Our scalable approach allows for learning high-capacity models that generalize well:* e.g., *a vanilla ViT-Huge model achieves the best accuracy (87.8%) among methods that use only ImageNet-1K data. Transfer performance in downstream tasks outperforms supervised pre-training and shows promising scaling behavior.*

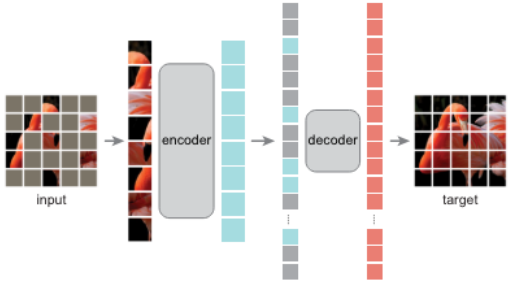


Figure 1. **Our MAE architecture**. During pre-training, a large random subset of image patches (*e.g.*, 75%) is masked out. The encoder is applied to the small subset of *visible patches*. Mask tokens are introduced *after* the encoder, and the full set of encoded patches and mask tokens is processed by a small decoder that reconstructs the original image in pixels. After pre-training, the decoder is discarded and the encoder is applied to uncorrupted images (full sets of patches) for recognition tasks.

in vision [59, 46] preceded BERT. However, despite significant interest in this idea following the success of BERT, progress of autoencoding methods in vision lags behind

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding

Jacob Devlin **Ming-Wei Chang** **Kenton Lee** **Kristina Toutanova**

Google AI Language

{jacobdevlin,mingweichang,kentonl,kristout}@google.com

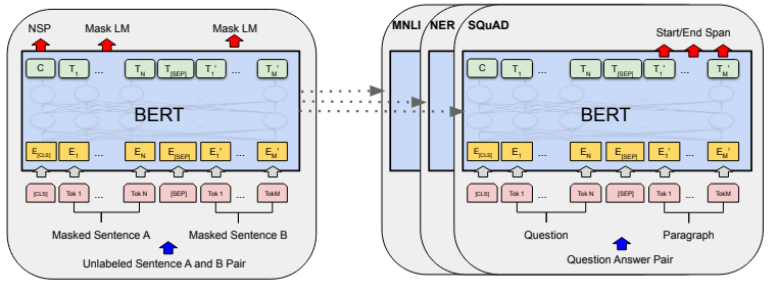
Abstract

We introduce a new language representation model called **BERT**, which stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers. Unlike recent language representation models (Peters et al., 2018a; Radford et al., 2018), BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications.

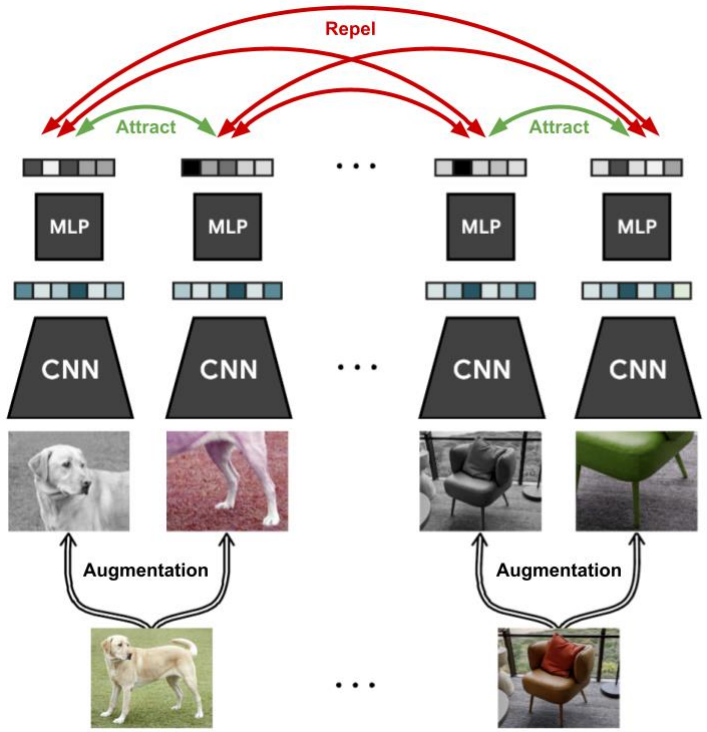
BERT is conceptually simple and empirically powerful. It obtains new state-of-the-art results on eleven natural language processing tasks, including pushing the GLUE score to 80.5% (7.7% point absolute improvement), MultiNLI accuracy to 86.7% (4.6% absolute improvement), SQuAD v1.1 question answering Test F1 to 93.2 (1.5 point absolute improvement) and SQuAD v2.0 Test F1 to 83.1 (5.1 point absolute improvement).

There are two existing strategies for applying pre-trained language representations to downstream tasks: *feature-based* and *fine-tuning*. The feature-based approach, such as ELMo (Peters et al., 2018a), uses task-specific architectures that include the pre-trained representations as additional features. The fine-tuning approach, such as the Generative Pre-trained Transformer (OpenAI GPT) (Radford et al., 2018), introduces minimal task-specific parameters, and is trained on the downstream tasks by simply fine-tuning *all* pre-trained parameters. The two approaches share the same objective function during pre-training, where they use unidirectional language models to learn general language representations.

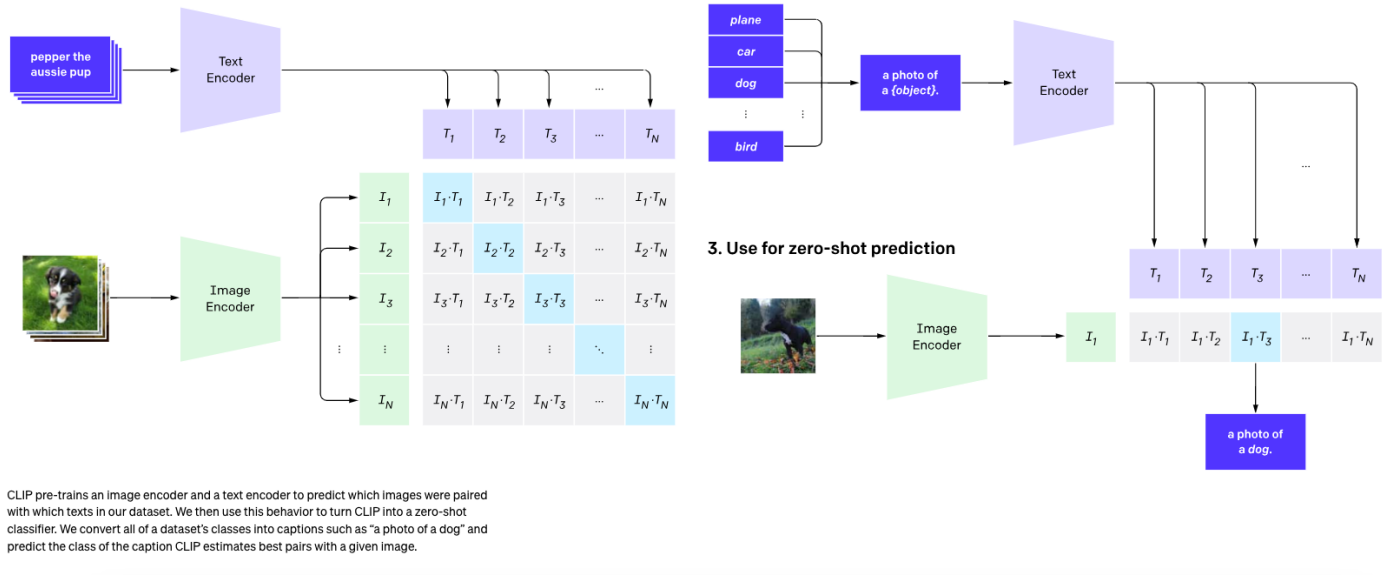
We argue that current techniques restrict the power of the pre-trained representations, especially for the fine-tuning approaches. The major limitation is that standard language models are unidirectional, and this limits the choice of architectures that can be used during pre-training. For example, in OpenAI GPT, the authors use a left-to-right architecture, where every token can only attend to previous tokens in the self-attention layers



Self-supervised Learning



https://simclr.github.io/



CLIP pre-trains an image encoder and a text encoder to predict which images were paired with which texts in our dataset. We then use this behavior to turn CLIP into a zero-shot classifier. We convert all of a dataset's classes into captions such as "a photo of a dog" and predict the class of the caption CLIP estimates best pairs with a given image.

https://openai.com/research/clip

Takeaways

- Generative vs discriminative modeling
- Generative Adversarial Neural Networks
- Autoencoders and Variational Autoencoders
- Masked Autoencoding
- Self-supervised Learning

References

- https://developers.google.com/machine-learning/gan/gan_structure
- https://developer.ibm.com/articles/generative-adversarial-networks-explained/
- https://lilianweng.github.io/posts/2018-10-13-flow-models/
- https://www.ibm.com/topics/autoencoder
- http://introtodeeplearning.com/2022/index.html