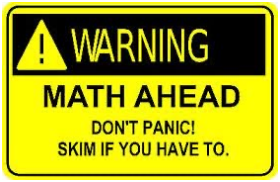


CSC413/2516 Lecture 10:

Generative Models & Graph Neural Networks

Jimmy Ba and Bo Wang

Logistics

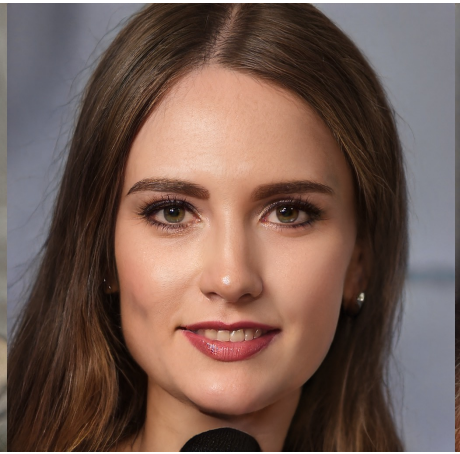
Some administrative stuff:

- HW4 is out! (Due on April 11, 40~50% less)
- HW3 is due March 31
- Grades for A1&A2, Comments for Proposals are released!

Overview

Quiz: Which face image is fake?

A



B

C

Overview

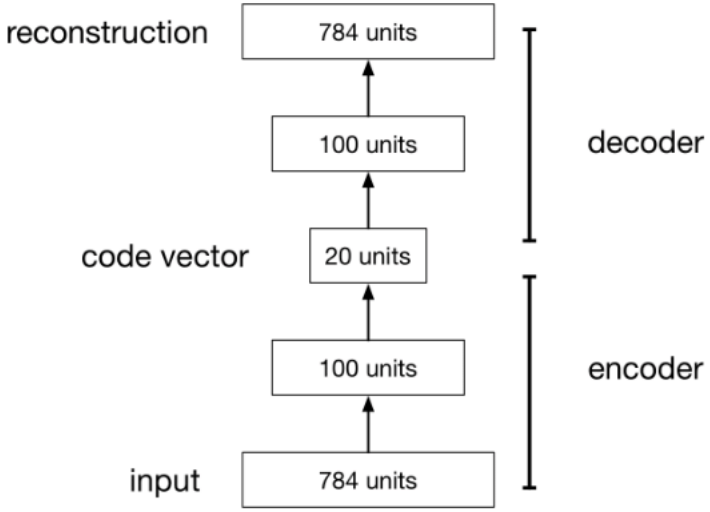
Four modern approaches to generative modeling:

- Autoregressive models (Lectures 3, 7, and 8)
- Generative adversarial networks (last lecture)
- LLMs (e.g., GPTs) (last lecture)
- Variational autoencoders (this lecture)

All four approaches have different pros and cons.

Autoencoders

- An autoencoder is a feed-forward neural net whose job it is to take an input **x** and predict **x**.
- To make this non-trivial, we need to add a bottleneck layer whose dimension is much smaller than the input.



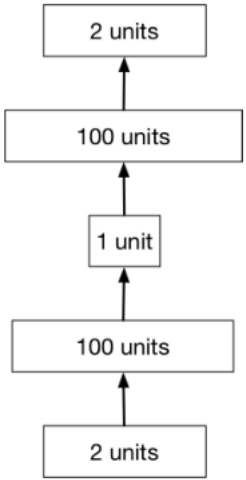
Autoencoders

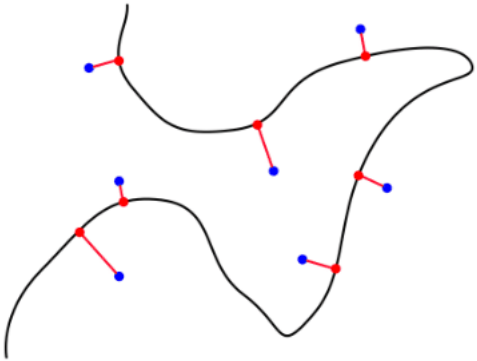
Why autoencoders?

- Map high-dimensional data to two dimensions for visualization
- Compression (i.e. reducing the file size)
  - Note: this requires a VAE, not just an ordinary autoencoder.
- Learn abstract features in an unsupervised way so you can apply them to a supervised task
  - Unlabled data can be much more plentiful than labeled data
- Learn a semantically meaningful representation where you can, e.g., interpolate between different images.

Deep Autoencoders

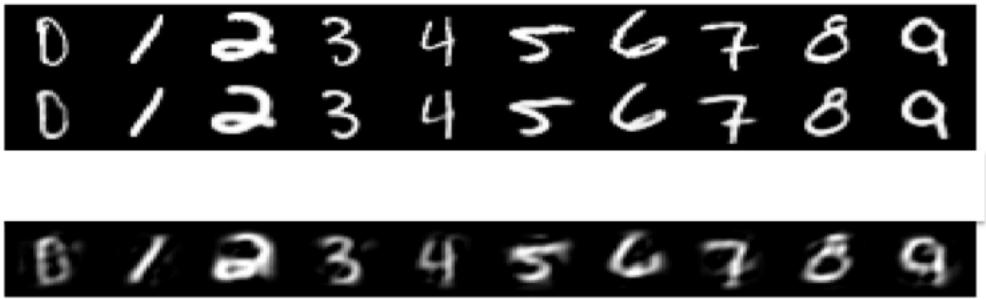
- Deep nonlinear autoencoders learn to project the data onto a low-dimensional nonlinear manifold.
- This manifold is the image of the decoder.
- This is a kind of nonlinear dimensionality reduction.





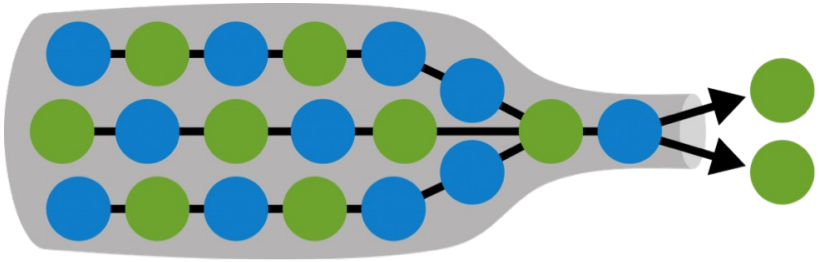
Deep Autoencoders

- Nonlinear autoencoders can learn more powerful codes for a given dimensionality, compared with linear autoencoders (PCA)

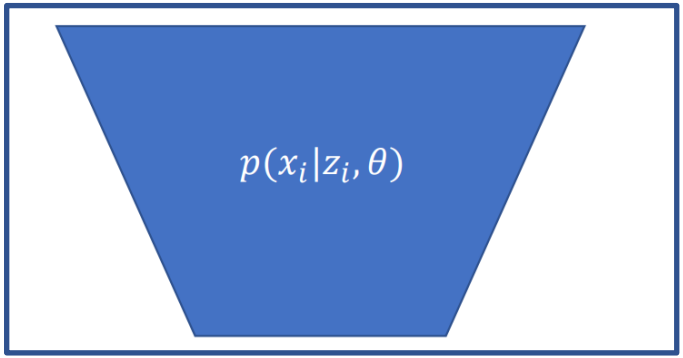


Deep Autoencoders

- Some limitations of autoencoders
  - They're not generative models, so they don't define a distribution
  - How to choose the latent dimension?



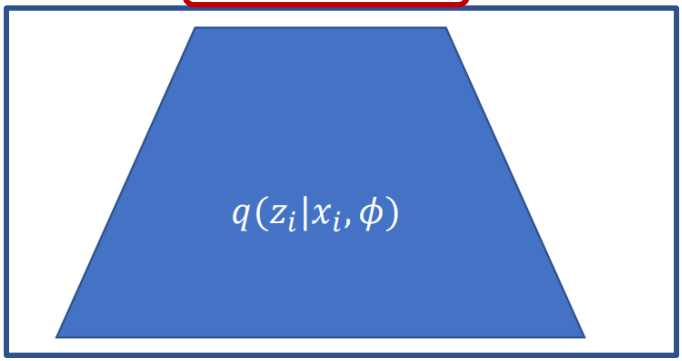
Variational Auto-encoder (VAE)



Decoder learns the generative process given the sampled latent vectors.

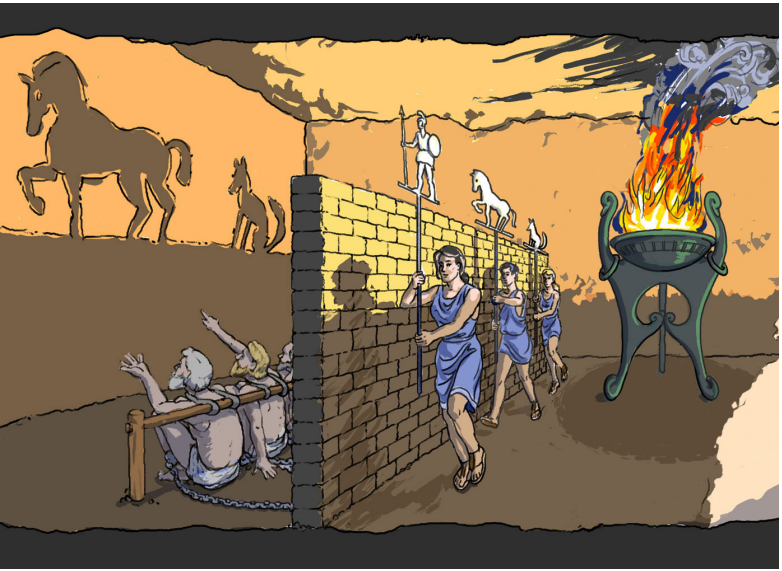
$z_i \sim q(z_i|x_i, \phi)$

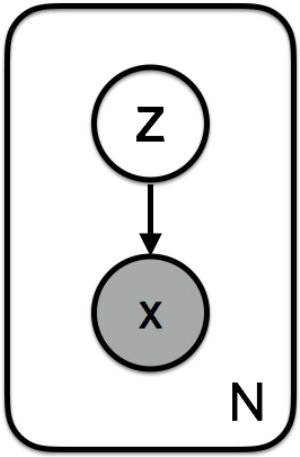
Sampling process in the middle.



Encoder learns the distribution of latent space given the observations.

Observation Model





Source: https://iagtm.pressbooks.com/chapter/story-platos-allegory-of-the-cave/

Observation Model

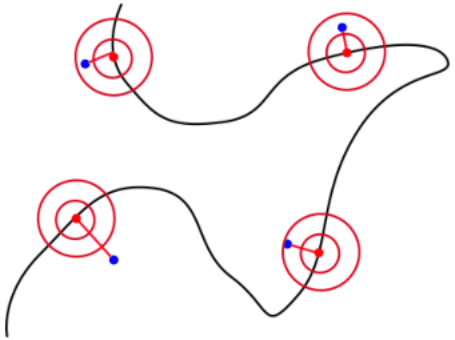
- Consider training a generator network with maximum likelihood.

$$p(\mathbf{x}) = \int p(\mathbf{z})p(\mathbf{x} \mid \mathbf{z}) \, \mathrm{d}\mathbf{z}$$

- One problem: if $\mathbf{z}$ is low-dimensional and the decoder is deterministic, then $p(\mathbf{x}) = 0$ almost everywhere!
  - The model only generates samples over a low-dimensional sub-manifold of $\mathcal{X}$.
- Solution: define a noisy observation model, e.g.

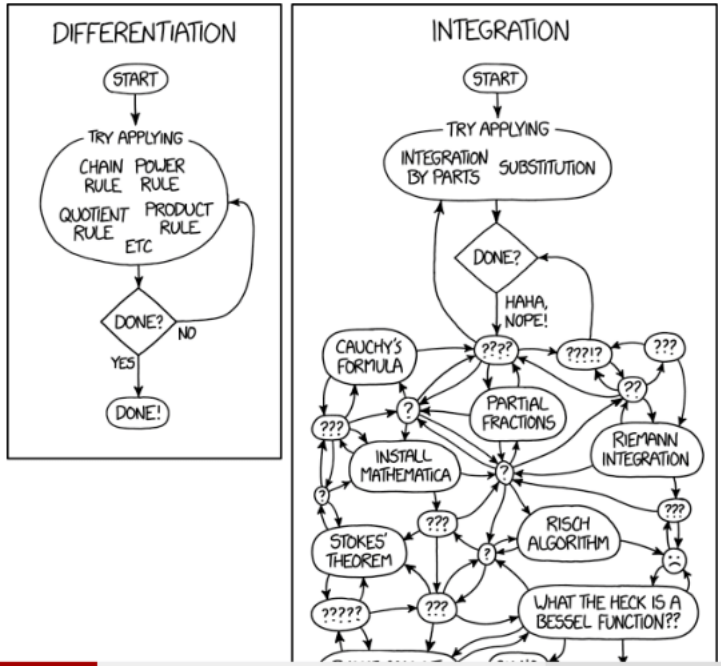
$$p(\mathbf{x} \mid \mathbf{z}) = \mathcal{N}(\mathbf{x}; G_{\boldsymbol{\theta}}(\mathbf{z}), \eta \mathbf{I}),$$

where $G_{\boldsymbol{\theta}}$ is the function computed by the decoder with parameters $\boldsymbol{\theta}$.



Observation Model

- At least $p(\mathbf{x}) = \int p(\mathbf{z})p(\mathbf{x} \mid \mathbf{z})\,\mathrm{d}\mathbf{z}$ is well-defined, but how can we compute it?
- Integration, according to XKCD:



Observation Model

- At least $p(\mathbf{x}) = \int p(\mathbf{z})p(\mathbf{x} \mid \mathbf{z}) \, \mathrm{d}\mathbf{z}$ is well-defined, but how can we compute it?
  - The decoder function $G_\theta(\mathbf{z})$ is very complicated, so there's no hope of finding a closed form.
- Instead, we will try to maximize a lower bound on $\log p(\mathbf{x})$.
  - The math is essentially the same as in the EM algorithm from CSC411.

Variational Inference

- We obtain the lower bound using Jensen's Inequality: for a convex function h of a random variable X,

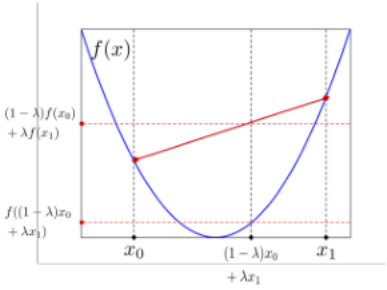
$$\mathbb{E}[h(X)] \geq h(\mathbb{E}[X])$$

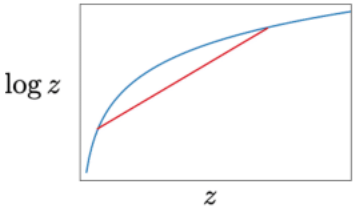
Therefore, if h is concave (i.e. $-h$ is convex),

$$\mathbb{E}[h(X)] \leq h(\mathbb{E}[X])$$

- The function $\log z$ is concave. Therefore,

$$\mathbb{E}[\log X] \leq \log \mathbb{E}[X]$$





Variational Inference

- Suppose we have some distribution $q(\mathbf{z})$. (We'll see later where this comes from.)
- We use Jensen's Inequality to obtain the lower bound.

$$\begin{aligned}
\log p(\mathbf{x}) &= \log \int p(\mathbf{z})\, p(\mathbf{x}|\mathbf{z})\, \mathrm{d}\mathbf{z} \\
&= \log \int q(\mathbf{z})\, \frac{p(\mathbf{z})}{q(\mathbf{z})} p(\mathbf{x}|\mathbf{z})\, \mathrm{d}\mathbf{z} \\
&\geq \int q(\mathbf{z}) \log \left[\frac{p(\mathbf{z})}{q(\mathbf{z})}\, p(\mathbf{x}|\mathbf{z}) \right] \mathrm{d}\mathbf{z} \\
&= \mathbb{E}_q \left[\log \frac{p(\mathbf{z})}{q(\mathbf{z})} \right] + \mathbb{E}_q \left[\log p(\mathbf{x}|\mathbf{z}) \right]
\end{aligned}$$

$$\mathbb{E}[\log X] \leq \log \mathbb{E}[X]$$
(Jensen's Inequality)

- We'll look at these two terms in turn.

Variational Inference

- The first term we'll look at is $\mathbb{E}_q \left[\log p(\mathbf{x}|\mathbf{z})\right]$
- Since we assumed a Gaussian observation model,

$$\begin{aligned}
\log p(\mathbf{x}|\mathbf{z}) &= \log \mathcal{N}(\mathbf{x}; G_{\boldsymbol{\theta}}(\mathbf{z}), \eta \mathbf{I}) \\
&= \log \left[\frac{1}{(2\pi\eta)^{D/2}} \exp\left(-\frac{1}{2\eta} \|\mathbf{x} - G_{\boldsymbol{\theta}}(\mathbf{z})\|^2 \right) \right] \\
&= -\frac{1}{2\eta} \|\mathbf{x} - G_{\boldsymbol{\theta}}(\mathbf{z})\|^2 + \text{const}
\end{aligned}$$

- So this term is the expected squared error in reconstructing $\mathbf{x}$ from $\mathbf{z}$. We call it the reconstruction term.

Variational Inference

- The second term is $\mathbb{E}_q \left[\log \frac{p(\mathbf{z})}{q(\mathbf{z})} \right]$.
- This is just $-\mathrm{D}_{\mathrm{KL}}(q(\mathbf{z}) \| p(\mathbf{z}))$, where D_{KL} is the Kullback-Leibler (KL) divergence

$$\mathrm{D}_{\mathrm{KL}}(q(\mathbf{z}) \| p(\mathbf{z})) \triangleq \mathbb{E}_q \left[\log \frac{q(\mathbf{z})}{p(\mathbf{z})} \right]$$

  - KL divergence is a widely used measure of distance between probability distributions, though it doesn't satisfy the axioms to be a distance metric.
  - More details in tutorial.
- Typically, $p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I})$. Hence, the KL term encourages q to be close to $\mathcal{N}(\mathbf{0}, \mathbf{I})$.

Variational Inference

- Hence, we're trying to maximize the variational lower bound, or variational free energy:

$$\log p(\mathbf{x}) \geq \mathcal{F}(\boldsymbol{\theta}, q) = \mathbb{E}_q \left[\log p(\mathbf{x}|\mathbf{z})\right] - \mathrm{D}_{\mathrm{KL}}(q\|p).$$

- The term "variational" is a historical accident: "variational inference" used to be done using variational calculus, but this isn't how we train VAEs.
- We'd like to choose q to make the bound as tight as possible.
- It's possible to show that the gap is given by:

Note: A good exercise for students!

$$\log p(\mathbf{x}) - \mathcal{F}(\boldsymbol{\theta}, q) = \mathrm{D}_{\mathrm{KL}}(q(\mathbf{z})\|p(\mathbf{z}|\mathbf{x})).$$

Therefore, we'd like q to be as close as possible to the posterior distribution $p(\mathbf{z}|\mathbf{x})$.

- Let's think about the role of each of the two terms.
- The reconstruction term

$$\mathbb{E}_q[\log p(\mathbf{x}|\mathbf{z})] = -\frac{1}{2\sigma^2}\mathbb{E}_q[\|\mathbf{x} - G_\theta(\mathbf{z})\|^2] + \text{const}$$

  is minimized when q is a point mass on

$$\mathbf{z}_* = \arg\min_{\mathbf{z}} \|\mathbf{x} - G_\theta(\mathbf{z})\|^2.$$

- But a point mass would have infinite KL divergence. (Exercise: check this.) So the KL term forces q to be more spread out.

Reparameterization Trick

- To fit q, let's assign it a parametric form, in particular a Gaussian distribution: $q(\mathbf{z}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$, where $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_K)$ and $\boldsymbol{\Sigma} = \text{diag}(\sigma_1^2, \ldots, \sigma_K^2)$.
- In general, it's hard to differentiate through an expectation. But for Gaussian q, we can apply the reparameterization trick:

$$z_i = \mu_i + \sigma_i \epsilon_i,$$

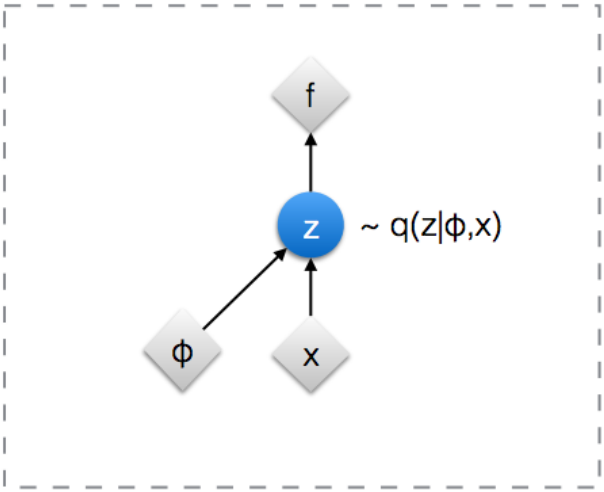
  where $\epsilon_i \sim \mathcal{N}(0, 1)$.
- Hence,

$$\overline{\mu_i} = \overline{z_i} \qquad \overline{\sigma_i} = \overline{z_i} \epsilon_i.$$

- This is exactly analogous to how we derived the backprop rules for dropout

Reparameterization Trick

Original form

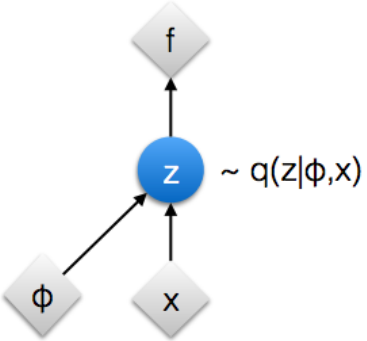


: Deterministic node

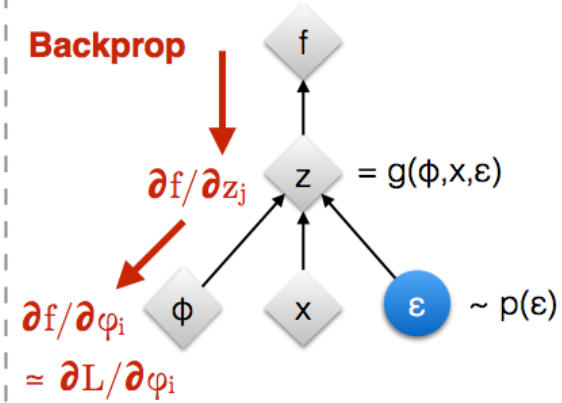
: Random node

Reparameterization Trick

Original form



Reparameterised form



: Deterministic node

: Random node

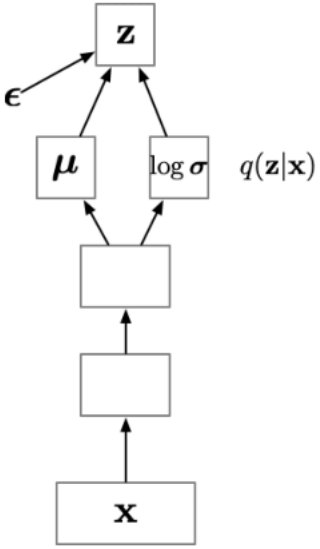
[Kingma, 2013]
[Bengio, 2013]
[Kingma and Welling 2014]
[Rezende et al 2014]

Amortization

- This suggests one strategy for learning the decoder. For each training example,
  1. Fit q to approximate the posterior for the current $\mathbf{x}$ by doing many steps of gradient ascent on $\mathcal{F}$.
  2. Update the decoder parameters $\boldsymbol{\theta}$ with gradient ascent on $\mathcal{F}$.
- **Problem:** this requires an expensive iterative procedure for every training example, so it will take a long time to process the whole training set.

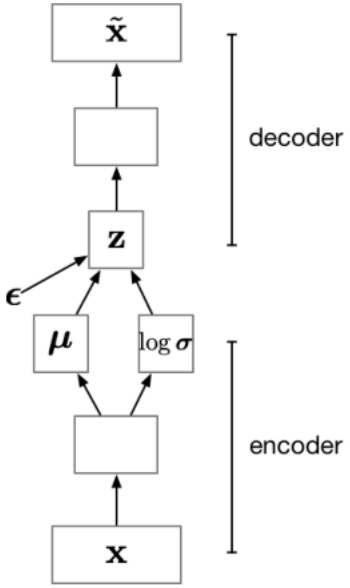
Amortization

- **Idea:** amortize the cost of inference by learning an inference network which predicts $(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ as a function of $\mathbf{x}$.
- The outputs of the inference net are $\boldsymbol{\mu}$ and $\log \boldsymbol{\sigma}$. (The log representation ensures $\boldsymbol{\sigma} > 0$.)
- If $\boldsymbol{\sigma} \approx \mathbf{0}$, then this network essentially computes $\mathbf{z}$ deterministically, by way of $\boldsymbol{\mu}$.
  - But the KL term encourages $\boldsymbol{\sigma} > 0$, so in general $\mathbf{z}$ will be noisy.
- The notation $q(\mathbf{z}|\mathbf{x})$ emphasizes that q depends on $\mathbf{x}$, even though it's not actually a conditional distribution.

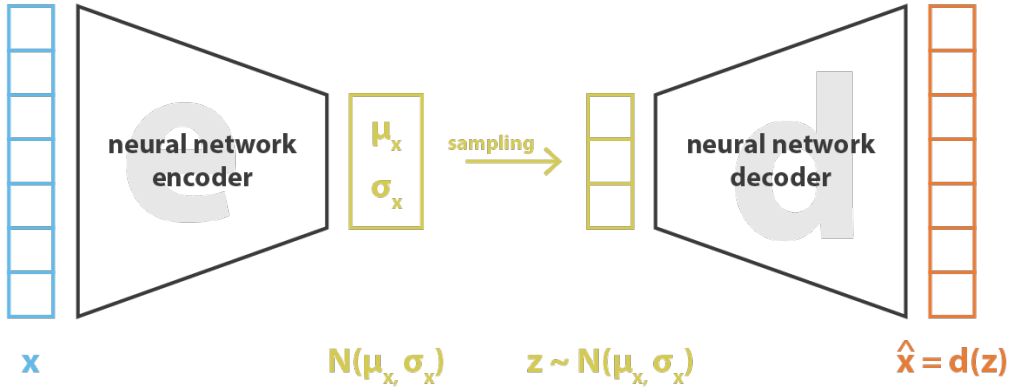


Amortization

- Combining this with the decoder network, we see the structure closely resembles an ordinary autoencoder. The inference net is like an encoder.
- Hence, this architecture is known as a variational autoencoder (VAE).
- The parameters of both the encoder and decoder networks are updated using a single pass of ordinary backprop.
  - The reconstruction term corresponds to squared error $\|\mathbf{x} - \tilde{\mathbf{x}}\|^2$, like in an ordinary VAE.
  - The KL term regularizes the representation by encouraging $\mathbf{z}$ to be more stochastic.



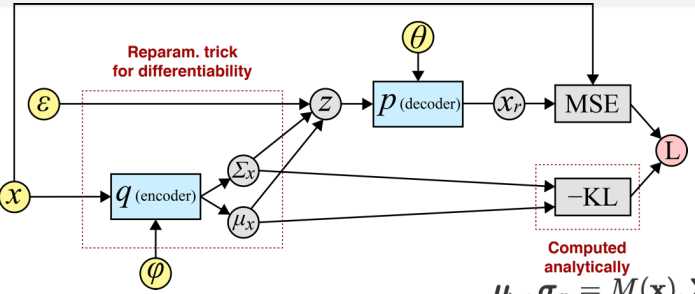
Variational Auto-encoder (VAE)



$$\text{loss} = ||x - \hat{x}||^2 + KL[N(\mu_x, \sigma_x), N(0, I)] = ||x - d(z)||^2 + KL[N(\mu_x, \sigma_x), N(0, I)]$$

Source: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73

VAE - Summary



$$\boldsymbol{\mu}_x, \boldsymbol{\sigma}_x = M(\mathbf{x}), \Sigma(\mathbf{x}) \qquad \text{Push } \mathbf{x} \text{ through encoder}$$

$$\boldsymbol{\epsilon} \sim \mathcal{N}(0, 1) \qquad \text{Sample noise}$$

$$\mathbf{z} = \boldsymbol{\epsilon}\boldsymbol{\sigma}_x + \boldsymbol{\mu}_x \qquad \text{Reparameterize}$$

$$\mathbf{x}_r = p_{\boldsymbol{\theta}}(\mathbf{x} \mid \mathbf{z}) \qquad \text{Push } \mathbf{z} \text{ through decoder}$$

$$\text{recon. loss} = \text{MSE}(\mathbf{x}, \mathbf{x}_r) \qquad \text{Compute reconstruction loss}$$

$$\text{var. loss} = -\text{KL}[\mathcal{N}(\boldsymbol{\mu}_x, \boldsymbol{\sigma}_x) \| \mathcal{N}(0, I)] \qquad \text{Compute variational loss}$$

$$\text{L} = \text{recon. loss} + \text{var. loss} \qquad \text{Combine losses}$$

Source: https://gregorygundersen.com/blog/2018/04/29/reparameterization/

VAEs vs. Other Generative Models

- In short, a VAE is like an autoencoder, except that it's also a generative model (defines a distribution $p(\mathbf{x})$).
- Unlike autoregressive models, generation only requires one forward pass.
- Unlike reversible models, we can fit a low-dimensional latent representation. We'll see we can do interesting things with this...

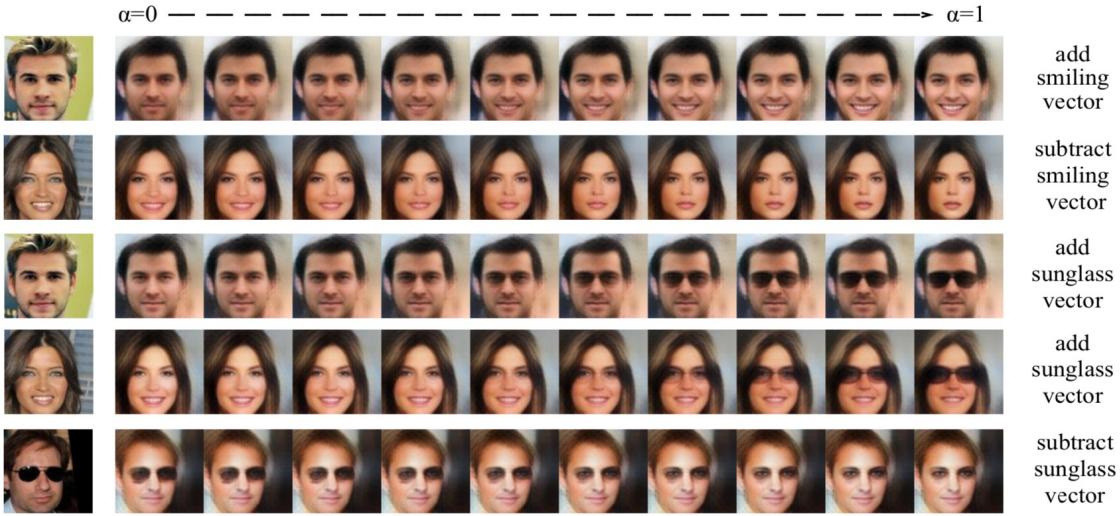
Latent Space Interpolations

- You can often get interesting results by interpolating between two vectors in the latent space:

Ha and Eck, "A neural representation of sketch drawings"

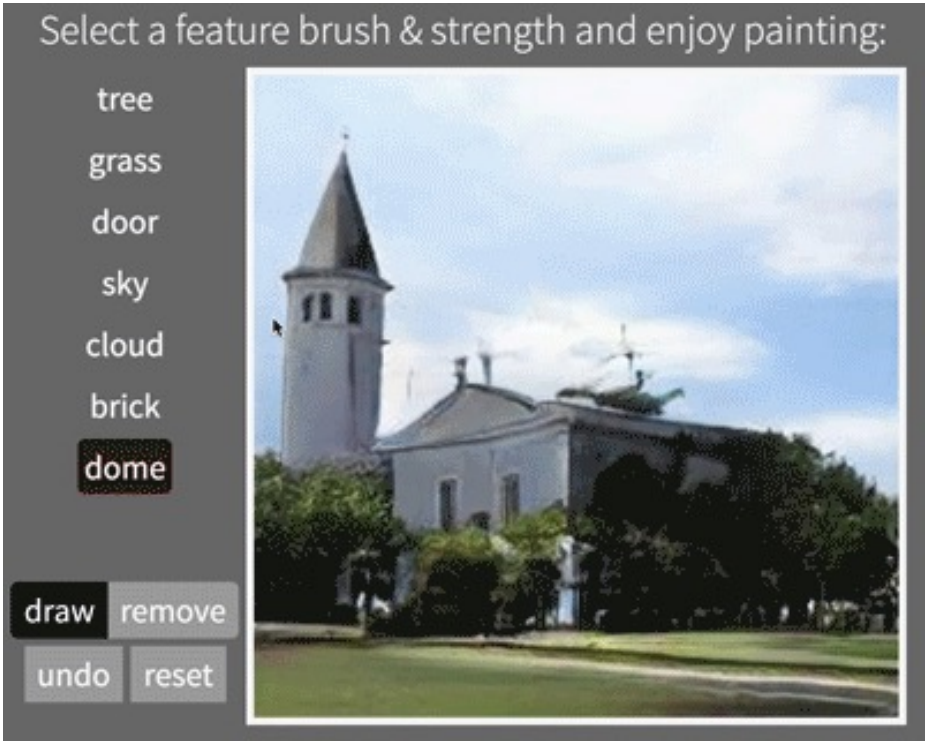
Latent Space Interpolations

- You can often get interesting results by interpolating between two vectors in the latent space:



https://arxiv.org/pdf/1610.00291.pdf

Latent Space Interpolations



Latent Space Interpolations

- Latent space interpolation of music:
  `https://magenta.tensorflow.org/music-vae`

After Break: Diffusion Models (optional)

Diffusion Probabilistic Models

Deep Unsupervised Learning using Nonequilibrium Thermodynamics

ICLR 2015

Jascha Sohl-Dickstein JASCHA@STANFORD.EDU
Stanford University

Eric A. Weiss EAWEISS@BERKELEY.EDU
University of California, Berkeley

Niru Maheswaranathan NIRUM@STANFORD.EDU
Stanford University

Surya Ganguli SGANGULI@STANFORD.EDU
Stanford University

Motivation: Estimating small perturbations is more tractable than explicitly describing the full distribution.

The essential idea, inspired by non-equilibrium statistical physics, is to

- systematically and slowly destroy structure in a data distribution through an iterative forward diffusion process.
- learn a reverse diffusion process that restores structure in data, yielding a highly flexible and tractable generative model of the data.

https://arxiv.org/abs/1503.03585
https://www.youtube.com/watch?v=XCUlnHP1TNM

Denoising Diffusion Probabilistic Model (DDPM)

Denoising Diffusion Probabilistic Models

Jonathan Ho
UC Berkeley
jonathanho@berkeley.edu

Ajay Jain
UC Berkeley
ajayj@berkeley.edu

Pieter Abbeel
UC Berkeley
pabbeel@cs.berkeley.edu

We present high quality image synthesis results using diffusion probabilistic models, a class of latent variable models inspired by considerations from nonequilibrium thermodynamics. Our best results are obtained by training on a weighted variational bound designed according to a novel connection between diffusion probabilistic models and denoising score matching with Langevin dynamics, and our models naturally admit a progressive lossy decompression scheme that can be interpreted as a generalization of autoregressive decoding. On the unconditional CIFAR10 dataset,

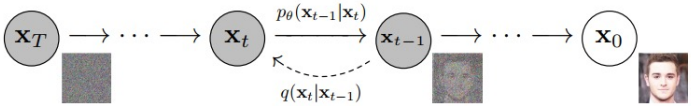


Figure 2: The directed graphical model considered in this work.

https://github.com/hojonathanho/diffusion

Ho et al., Denoising Diffusion Probabilistic Models, NeurIPS 2020

Demonstrate that diffusion models are capable of generating high quality samples.

Denoising Diffusion Probabilistic Model (DDPM)

Forward diffusion process: gradually adds noise to input image $q(x_t|x_{t-1})$

Reverse denoising process: learns to generate data by denoising $q(x_{t-1}|x_t) \approx p_\theta(x_{t-1}|x_t)$

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

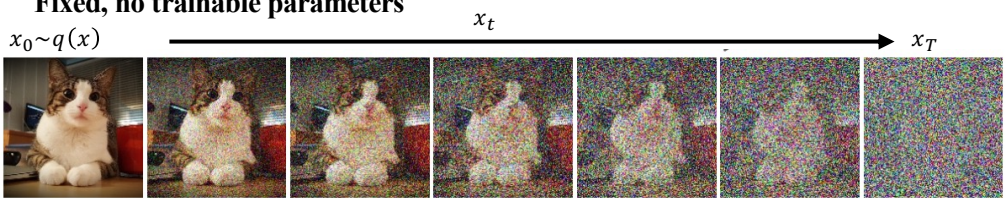
How are these formulas derived?

Ho et al., Denoising Diffusion Probabilistic Models, NeurIPS 2020

Denoising Diffusion Probabilistic Model (DDPM)

DDPM: Forward Process $q(x_t|x_{t-1})$

Fixed, no trainable parameters



Q: How to obtain x_t at any arbitrary time step t? $\beta_0 = 10^{-4}, \beta_T = 0.02$

Def: $q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$ $\quad q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1})$

Reparameterization trick
A way to sample data x from $N(\mu, \sigma^2)$
- Sample z from $N(0,1)$
- $x = \mu + \sigma z$

A nice property of the above process is that we can sample $\mathbf{x}_t$ at any arbitrary time step t in a closed form using reparameterization trick. Let $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \prod_{i=1}^{T} \alpha_i$:

$$\begin{aligned}
\mathbf{x}_t &= \sqrt{\alpha_t}\mathbf{x}_{t-1} + \sqrt{1-\alpha_t}\mathbf{z}_{t-1} && ;\text{where } \mathbf{z}_{t-1}, \mathbf{z}_{t-2}, \cdots \sim \mathcal{N}(\mathbf{0}, \mathbf{I}) \\
&= \sqrt{\alpha_t\alpha_{t-1}}\mathbf{x}_{t-2} + \sqrt{1-\alpha_t\alpha_{t-1}}\bar{\mathbf{z}}_{t-2} && ;\text{where } \bar{\mathbf{z}}_{t-2} \text{ merges two Gaussians (*).} \\
&= \dots \\
&= \sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\mathbf{z} \\
q(\mathbf{x}_t|\mathbf{x}_0) &= \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I})
\end{aligned}$$

(*) Recall that when we merge two Gaussians with different variance, $\mathcal{N}(\mathbf{0}, \sigma_1^2\mathbf{I})$ and $\mathcal{N}(\mathbf{0}, \sigma_2^2\mathbf{I})$, the new distribution is $\mathcal{N}(\mathbf{0}, (\sigma_1^2 + \sigma_2^2)\mathbf{I})$. Here the merged standard deviation is $\sqrt{(1-\alpha_t) + \alpha_t(1-\alpha_{t-1})} = \sqrt{1-\alpha_t\alpha_{t-1}}$.

DDPM: Training Process

Clean images

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1 - \bar{\alpha}_t} z$$

$$t \sim U(\{1, \ldots, T\})$$

Noisy images

U-Net

Predicted noise $\tilde{z}$

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \ldots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

$$\text{Loss}\ (z, \tilde{z}(x_t, t))$$

DDPM: An alternative way to derive the loss

Training is performed by optimizing the usual variational bound on negative log likelihood:

$$\mathbb{E}\left[-\log p_\theta(\mathbf{x}_0)\right] \leq \mathbb{E}_q\left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right] = \mathbb{E}_q\left[-\log p(\mathbf{x}_T) - \sum_{t>1} \log \frac{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}{q(\mathbf{x}_t|\mathbf{x}_{t-1})}\right] =: L \quad (3)$$

Efficient training is therefore possible by optimizing random terms of L with stochastic gradient descent. Further improvements come from variance reduction by rewriting L (3) as:

$$\mathbb{E}_q\Big[\underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_T|\mathbf{x}_0) \parallel p(\mathbf{x}_T))}_{L_T} + \sum_{t>1} \underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \parallel p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}} \underbrace{- \log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0}\Big] \quad (5)$$

Furthermore, with the parameterization (11), Eq. (10) simplifies to:

$$\mathbb{E}_{\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\frac{\beta_t^2}{2\sigma_t^2\alpha_t(1-\bar{\alpha}_t)}\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t)\right\|^2\right]$$

$$\begin{aligned}
L_{\mathrm{CE}} &= -\mathbb{E}_{q(\mathbf{x}_0)} \log p_\theta(\mathbf{x}_0) \\
&= -\mathbb{E}_{q(\mathbf{x}_0)} \log \left(\int p_\theta(\mathbf{x}_{0:T}) d\mathbf{x}_{1:T} \right) \\
&= -\mathbb{E}_{q(\mathbf{x}_0)} \log \left(\int q(\mathbf{x}_{1:T}|\mathbf{x}_0) \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} d\mathbf{x}_{1:T} \right) \\
&= -\mathbb{E}_{q(\mathbf{x}_0)} \log \left(\mathbb{E}_{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \right) \\
&\leq -\mathbb{E}_{q(\mathbf{x}_{0:T})} \log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)} \\
&= \mathbb{E}_{q(\mathbf{x}_{0:T})}\left[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\right] = L_{\mathrm{VLB}}
\end{aligned}$$

Jensen inequality

$$\begin{aligned}
L_{\mathrm{VLB}} &= \mathbb{E}_{q(\mathbf{x}_{0:T})}\left[\log \frac{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}{p_\theta(\mathbf{x}_{0:T})}\right] \\
&= \mathbb{E}_q\left[\log \frac{\prod_{t=1}^T q(\mathbf{x}_t|\mathbf{x}_{t-1})}{p_\theta(\mathbf{x}_T)\prod_{t=1}^T p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}\right] \\
&= \mathbb{E}_q\left[-\log p_\theta(\mathbf{x}_T) + \sum_{t=1}^T \log \frac{q(\mathbf{x}_t|\mathbf{x}_{t-1})}{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)}\right] \\
&= \mathbb{E}_q\left[-\log p_\theta(\mathbf{x}_T) + \sum_{t=2}^T \log \frac{q(\mathbf{x}_t|\mathbf{x}_{t-1})}{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)} + \log \frac{q(\mathbf{x}_1|\mathbf{x}_0)}{p_\theta(\mathbf{x}_0|\mathbf{x}_1)}\right] \\
&= \mathbb{E}_q\left[-\log p_\theta(\mathbf{x}_T) + \sum_{t=2}^T \log \left(\frac{q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)}{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)} \cdot \frac{q(\mathbf{x}_t|\mathbf{x}_0)}{q(\mathbf{x}_{t-1}|\mathbf{x}_0)}\right) + \log \frac{q(\mathbf{x}_1|\mathbf{x}_0)}{p_\theta(\mathbf{x}_0|\mathbf{x}_1)}\right] \\
&= \mathbb{E}_q\left[-\log p_\theta(\mathbf{x}_T) + \sum_{t=2}^T \log \frac{q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)}{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)} + \sum_{t=2}^T \log \frac{q(\mathbf{x}_t|\mathbf{x}_0)}{q(\mathbf{x}_{t-1}|\mathbf{x}_0)} + \log \frac{q(\mathbf{x}_1|\mathbf{x}_0)}{p_\theta(\mathbf{x}_0|\mathbf{x}_1)}\right] \\
&= \mathbb{E}_q\left[-\log p_\theta(\mathbf{x}_T) + \sum_{t=2}^T \log \frac{q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)}{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)} + \log \frac{q(\mathbf{x}_T|\mathbf{x}_0)}{q(\mathbf{x}_1|\mathbf{x}_0)} + \log \frac{q(\mathbf{x}_1|\mathbf{x}_0)}{p_\theta(\mathbf{x}_0|\mathbf{x}_1)}\right] \\
&= \mathbb{E}_q\left[\log \frac{q(\mathbf{x}_T|\mathbf{x}_0)}{p_\theta(\mathbf{x}_T)} + \sum_{t=2}^T \log \frac{q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)}{p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)} - \log p_\theta(\mathbf{x}_0|\mathbf{x}_1)\right] \\
&= \mathbb{E}_q[\underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_T|\mathbf{x}_0) \parallel p_\theta(\mathbf{x}_T))}_{L_T} + \sum_{t=2}^T \underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) \parallel p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}} \underbrace{- \log p_\theta(\mathbf{x}_0|\mathbf{x}_1)}_{L_0}]
\end{aligned}$$

Monster comes…

Separate case

Bayes' Rule

DDPM: Reverse Process $q(x_{t-1}|x_t)$

Reverse denoising process: learns to generate data by denoising $q(x_{t-1}|x_t) \approx p_\theta(x_{t-1}|x_t)$

$x_0 \sim q(x)$  x_T

Q2: How to estimate the true reverse process $q(x_{t-1}|x_t)$?

Remark 1. If the variance is small enough during forward process, $q(x_{t-1}|x_t)$ will be Gaussian as well.

Remark 2. The reverse conditional probability $q(x_{t-1}|x_t)$ is intractable, but $q(x_{t-1}|x_t, x_0)$ would be tractable.

Reminder: Gaussian pdf: $N(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$

DDPM: Reverse Process $q(x_{t-1}|x_t)$

Q3: How to obtain the mean and variance of $q(x_{t-1}|x_t)$?

It is noteworthy that the reverse conditional probability is tractable when conditioned on $\mathbf{x}_0$:

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1}; \tilde{\boldsymbol{\mu}}(\mathbf{x}_t, \mathbf{x}_0), \tilde{\beta}_t \mathbf{I})$$

Using Bayes' rule, we have:

$$\begin{aligned}
q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0) &= q(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{x}_0)\frac{q(\mathbf{x}_{t-1}|\mathbf{x}_0)}{q(\mathbf{x}_t|\mathbf{x}_0)} \\
&\propto \exp\Big(-\frac{1}{2}\big(\frac{(\mathbf{x}_t - \sqrt{\alpha_t}\mathbf{x}_{t-1})^2}{\beta_t} + \frac{(\mathbf{x}_{t-1} - \sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0)^2}{1-\bar{\alpha}_{t-1}} - \frac{(\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0)^2}{1-\bar{\alpha}_t}\big)\Big) \\
&= \exp\Big(-\frac{1}{2}\big(\frac{\mathbf{x}_t^2 - 2\sqrt{\alpha_t}\mathbf{x}_t\mathbf{x}_{t-1} + \alpha_t\mathbf{x}_{t-1}^2}{\beta_t} + \frac{\mathbf{x}_{t-1}^2 - 2\sqrt{\bar{\alpha}_{t-1}}\mathbf{x}_0\mathbf{x}_{t-1} + \bar{\alpha}_{t-1}\mathbf{x}_0^2}{1-\bar{\alpha}_{t-1}} - \frac{(\mathbf{x}_t - \sqrt{\bar{\alpha}_t}\mathbf{x}_0)^2}{1-\bar{\alpha}_t}\big)\Big) \\
&= \exp\Big(-\frac{1}{2}\big((\frac{\alpha_t}{\beta_t} + \frac{1}{1-\bar{\alpha}_{t-1}})\mathbf{x}_{t-1}^2 - (\frac{2\sqrt{\alpha_t}}{\beta_t}\mathbf{x}_t + \frac{2\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_{t-1}}\mathbf{x}_0)\mathbf{x}_{t-1} + C(\mathbf{x}_t, \mathbf{x}_0)\big)\Big)
\end{aligned}$$

Denoising Diffusion Probabilistic Model (DDPM) (optional)

DDPM: Reverse Process $q(x_{t-1}|x_t)$

Q3: How to obtain the mean and variance of $q(x_t|x_{t-1})$?

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)=\exp\Big(-\frac{1}{2}\big((\frac{\alpha_t}{\beta_t}+\frac{1}{1-\bar{\alpha}_{t-1}})\mathbf{x}_{t-1}^2-(\frac{2\sqrt{\alpha_t}}{\beta_t}\mathbf{x}_t+\frac{2\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_{t-1}}\mathbf{x}_0)\mathbf{x}_{t-1}+C(\mathbf{x}_t,\mathbf{x}_0)\big)\Big)$$

Following the standard Gaussian density function, the mean and variance can be parameterized as follows

$$\tilde{\beta}_t=1/(\frac{\alpha_t}{\beta_t}+\frac{1}{1-\bar{\alpha}_{t-1}})=1/(\frac{\alpha_t-\bar{\alpha}_t+\beta_t}{\beta_t(1-\bar{\alpha}_{t-1})})=\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\cdot\beta_t$$

Variance: $\tilde{\beta}_t=\frac{1}{a}$

$$\tilde{\boldsymbol{\mu}}_t(\mathbf{x}_t,\mathbf{x}_0)=(\frac{\sqrt{\alpha_t}}{\beta_t}\mathbf{x}_t+\frac{\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_{t-1}}\mathbf{x}_0)/(\frac{\alpha_t}{\beta_t}+\frac{1}{1-\bar{\alpha}_{t-1}})$$

$$=(\frac{\sqrt{\alpha_t}}{\beta_t}\mathbf{x}_t+\frac{\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_{t-1}}\mathbf{x}_0)\frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\cdot\beta_t$$

$$=\frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t+\frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0$$

Mean: $\tilde{\mu}(x_t,x_0)=-\frac{b}{2a}$

Using the fact that $ax^2+bx+C=a(x+\frac{b}{2a})^2$, rearrange the second line with regards to x_{t-1}

- x_{t-1}^2: $\frac{\alpha_t}{\beta_t}x_{t-1}^2+\frac{1}{1-\bar{\alpha}_{t-1}}x_{t-1}^2$, so $a=\frac{\alpha_t}{\beta_t}+\frac{1}{1-\bar{\alpha}_{t-1}}$.
- x_{t-1}: $(-\frac{2\sqrt{\alpha_t}}{\beta_t}x_t)x_{t-1}+(-\frac{2\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_{t-1}}x_0)x_{t-1}$, so $b=-(\frac{2\sqrt{\alpha_t}}{\beta_t}x_t+\frac{2\sqrt{\bar{\alpha}_{t-1}}}{1-\bar{\alpha}_{t-1}}x_0)$

Absorb x_0 by substituting it with

$$x_0=\frac{1}{\sqrt{a_t}}(x_t-\sqrt{1-a_t}z_t)$$

$$\tilde{\mu}_t=\frac{1}{\sqrt{a_t}}\Big(x_t-\frac{\beta_t}{\sqrt{1-a_t}}z_t\Big)$$

DDPM: Reverse Process $q(x_{t-1}|x_t)$

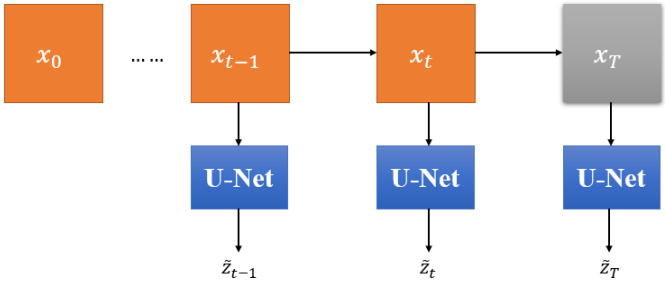
$$q(x_{t-1}|x_t) \sim N\left(\frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}} z_t\right), \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t I\right)$$

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\alpha_t}} z_t\right) + \sigma z$$

$$z \sim N(0, I), z_t \approx \tilde{z} = UNet(x_t, t)$$

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \ldots, 1$ **do**
3: $\quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\quad \mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$



DDPM: Summary

Forward

$$q(x_t|x_{t-1}) = N(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I)$$

$$q(x_t|x_0) = N(x_t; \sqrt{\bar{\alpha}_t}x_0, (1-\bar{\alpha}_t)I)$$

$$x_t = \sqrt{\bar{\alpha}_t}x_0 + (1-\bar{\alpha}_t)z,\ z \sim N(0,1)$$

$$\tilde{z} = UNet(x_t, t)$$

$$loss(z, \tilde{z})$$

Reverse $x_{t-1} \sim N(\mu, \sigma^2)$

$q(x_{t-1}|x_t) \rightarrow q(x_{t-1}|x_t, x_0) \rightarrow \frac{q(x_t|x_{t-1})q(x_{t-1}|x_0)}{q(x_t|x_0)}$ →derive mean and variance

$$q(x_{t-1}|x_t) = N\left(\frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}z_t\right), \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t I\right)$$

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}z_t\right) + \sigma z$$

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

DDPM: Sample Quality

Figure 3: LSUN Church samples. FID=7.89

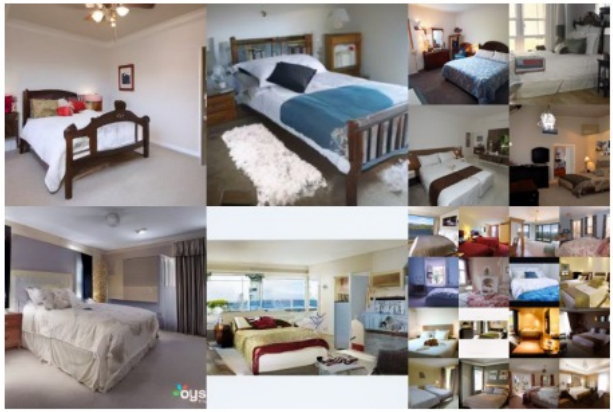


Figure 4: LSUN Bedroom samples. FID=4.90

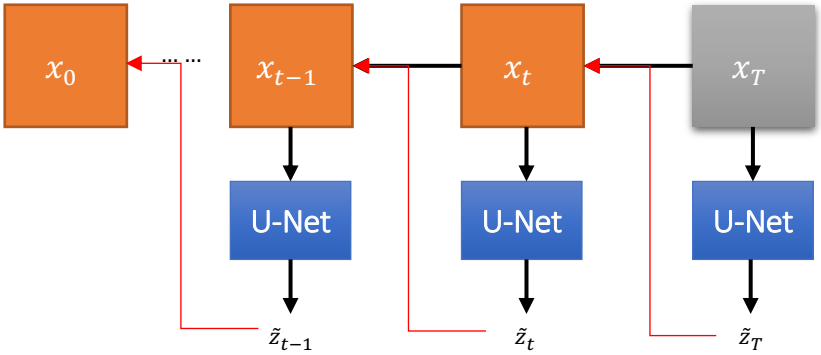
We find that training our models on the true variational bound yields better codelengths than training on the simplified objective, as expected, but the latter yields the best sample quality.

DDPM: Progressive generation

Figure 6: Unconditional CIFAR10 progressive generation ($\hat{\mathbf{x}}_0$ over time, from left to right). Extended samples and sample quality metrics over time in the appendix (Figs. 10 and 14).

Large scale image features appear first and details appear last.

DDPM: Reverse Process



$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t} z_t$$

Reverse process: $(x_t, \tilde{z}) \rightarrow x_{t-1}$

$$x_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(x_t - \sqrt{1-\bar{\alpha}_t} z_t)$$

One step can recover x_0!
Why don't we use it?

Denoising Diffusion Probabilistic Model (DDPM)

Denoising Diffusion Probabilistic Model (DDPM)

DDPM: $x_t \to x_0$

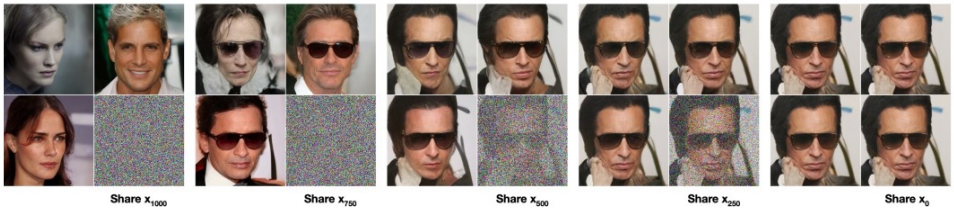


Figure 7: When conditioned on the same latent, CelebA-HQ 256×256 samples share high-level attributes. Bottom-right quadrants are $\mathbf{x}_t$, and other quadrants are samples from $p_\theta(\mathbf{x}_0|\mathbf{x}_t)$.

appear first and details appear last. Figure 7 shows stochastic predictions $\mathbf{x}_0 \sim p_\theta(\mathbf{x}_0|\mathbf{x}_t)$ with $\mathbf{x}_t$ frozen for various t. When t is small, all but fine details are preserved, and when t is large, only large scale features are preserved. Perhaps these are hints of conceptual compression [18].

$$x_t = \sqrt{\bar{\alpha}_t} x_0 + \sqrt{1-\bar{\alpha}_t} z_t$$

Reverse process: $(x_t, \tilde{z}) \to x_{t-1}$

$$x_0 = \frac{1}{\sqrt{\bar{\alpha}_t}}(x_t - \sqrt{1-\bar{\alpha}_t} z_t)$$

One step can recover x_0!
Why don't we use it?

Demo Time: Stable Diffusion

https://stablediffusionweb.com/

Trade-offs of Generative Approaches

- So far, we have seen four different approaches:
  - Autoregressive models (Lectures 3, 7, and 8)
  - Generative adversarial networks (last lecture)
  - Variational Auto-encoder (this lecture)
  - Diffusion models (this lecture)
- They all have their own pro and con. We often pick a method based on our application needs.
- Some considerations for computer vision applications:
  - Do we need to evaluate log likelihood of new data?
  - Do we prefer good samples over evaluation metric?
  - How imporant is representation learning, i.e. meaningful code vectors?
  - How much computational resource can we spent?

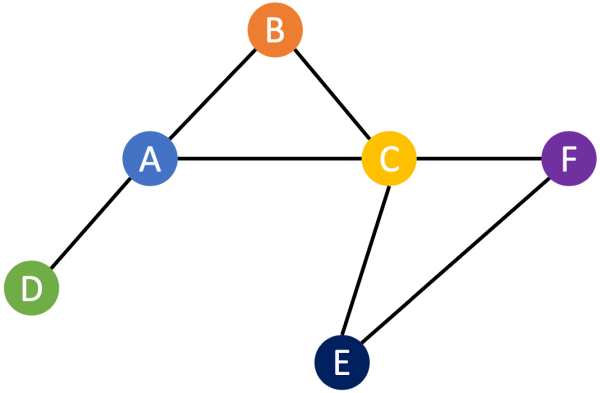
After the break

After the break: **Graph Neural Networks (GNN)**

The missing piece

- Tabular data : Linear Models, MLP
- Sequence data (e.g., Language, speech): CNN, RNN, Transformer
- Imaging data : CNN, Vision Transformer
- What about ***graph*** data?

What is a graph?



A graph is composed of

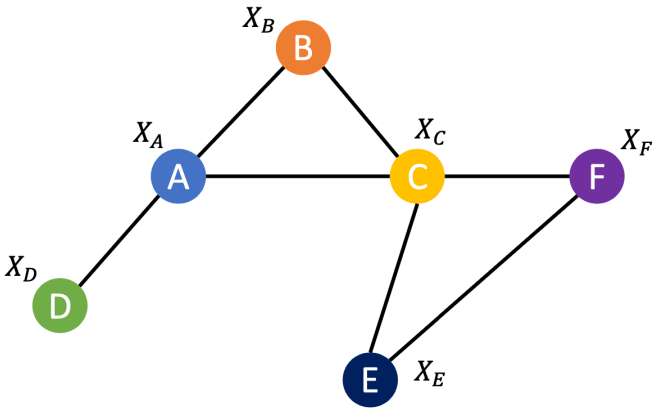
- **Nodes** (also called vertices)
- **Edges** connecting a pair of nodes

presented in an **adjacency matrix**

	A	B	C	D	E	F
A		1	1	1		
B	1		1			
C	1	1			1	1
D	1					
E			1			1
F			1		1	

Source: Minji Yoon, CMU

What is a graph?



A graph is composed of

- **Nodes** (also called vertices)
- **Edges** connecting a pair of nodes

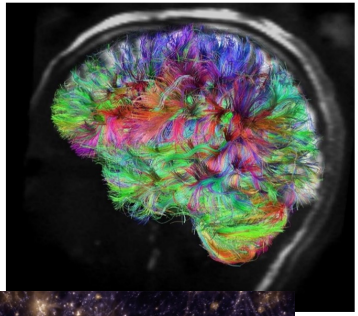
presented in an **adjacency matrix**

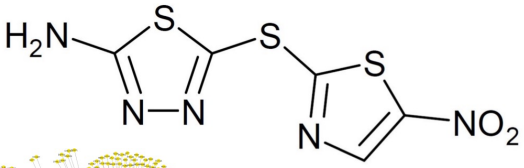
Nodes can have **feature vectors**

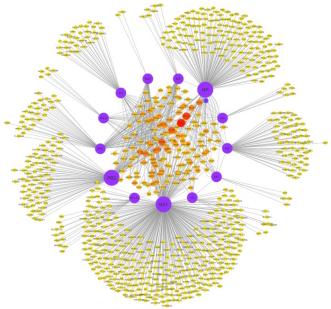
A	X_A
B	X_B
C	X_C
D	X_D
E	X_E
F	X_F

Source: Minji Yoon, CMU

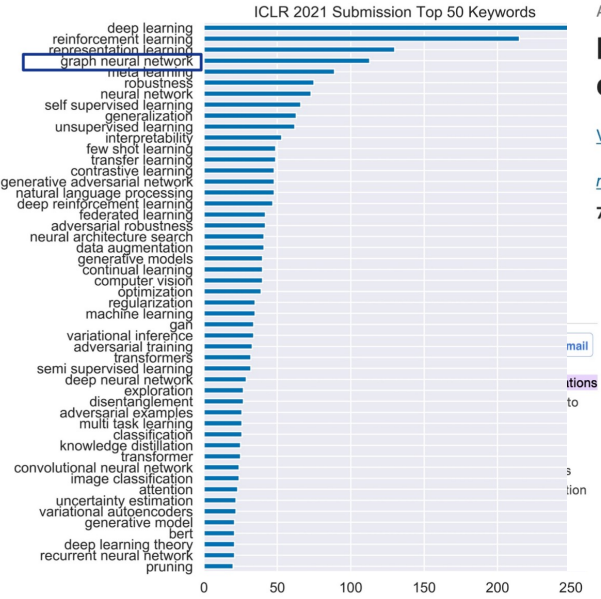
Graph is everywhere!







Graph Neural Networks (GNN) is everywhere



npj | computational materials

Explore content · About the journal · Publish with us

nature › npj computational materials › articles › article

Article | Open Access | Published: 03 June 2021

Benchmarking graph neural networks for materials chemistry

Victor Fung, Jiaxin Zhang, Eric Juarez & Bobby G. Sumpter

npj Computational Materials **7**, Article number: 84 (2021) | Cite this article

7807 Accesses | **7** Citations | **41** Altmetric | Metrics

nature

View all journals · Search · Login

Explore content · About the journal · Publish with us

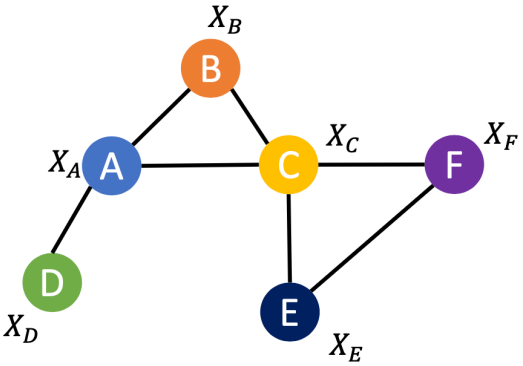
nature › articles › article

Article | Published: 09 June 2021

A graph placement methodology for fast chip design

Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Quoc V. Le, James Laudon, Richard Ho, Roger Carpenter & Jeff Dean

What is GNN? – Problem Setup

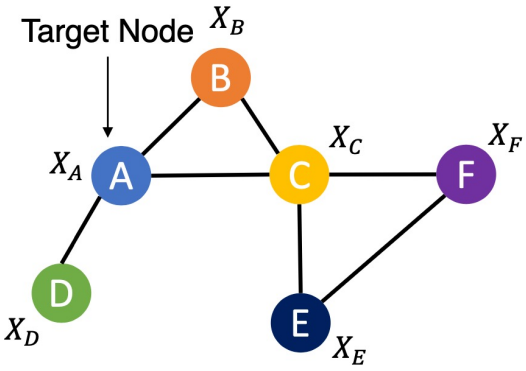


- **Given**
  - A graph
  - Node attributes
  - (part of nodes are labeled)
- **Find**
  - Node embeddings
- **Predict**
  - Labels for the remaining nodes

undirected unweighted graph

Source: Minji Yoon, CMU

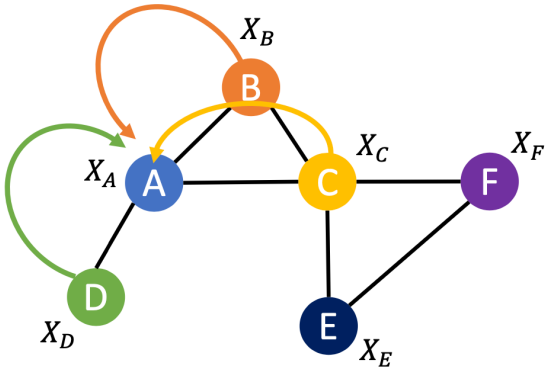
What is GNN? – Problem Setup



"Homophily: connected nodes are related/informative/similar"

Source: Minji Yoon, CMU

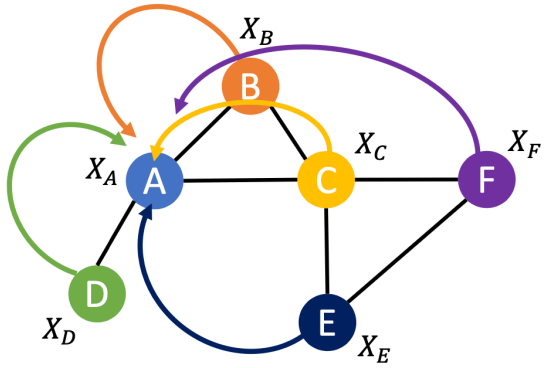
What is GNN? – Problem Setup



"Homophily: connected nodes are related/informative/similar"

Source: Minji Yoon, CMU

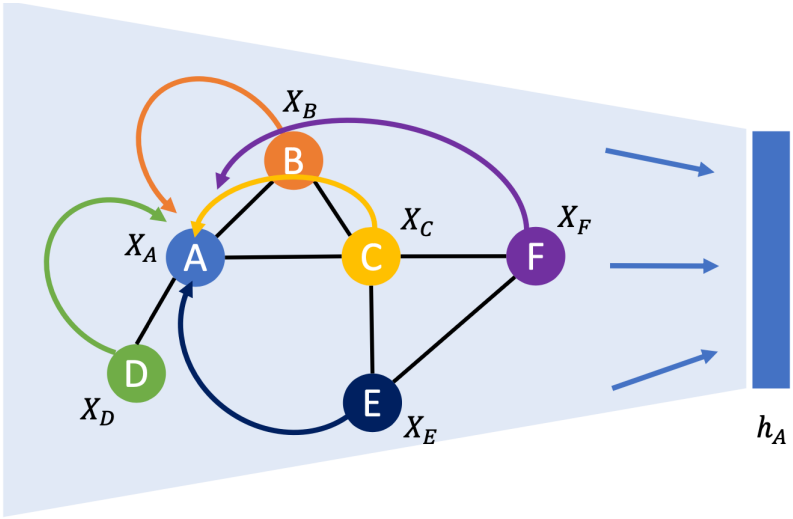
What is GNN? – Problem Setup



"Homophily: connected nodes are related/informative/similar"

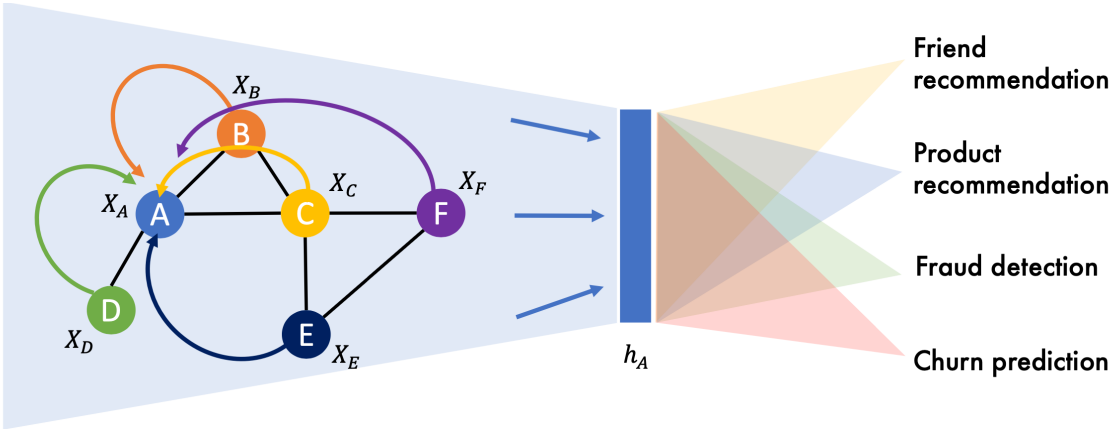
Source: Minji Yoon, CMU

What is GNN? – Problem Setup



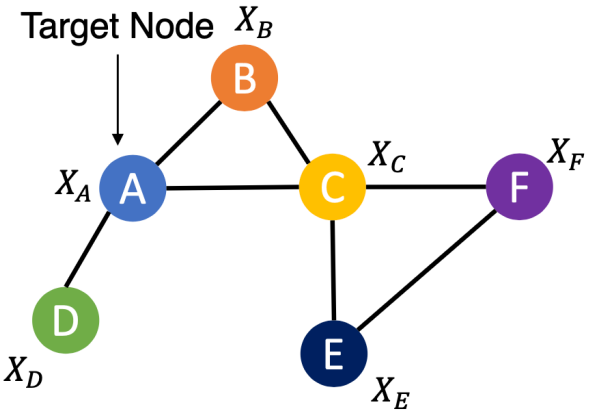
Source: Minji Yoon, CMU

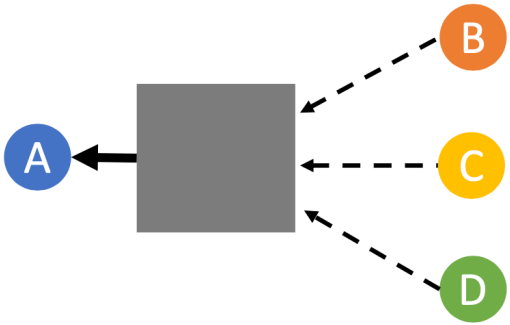
What is GNN? – Problem Setup



Source: Minji Yoon, CMU

What is GNN? – Forward propagation

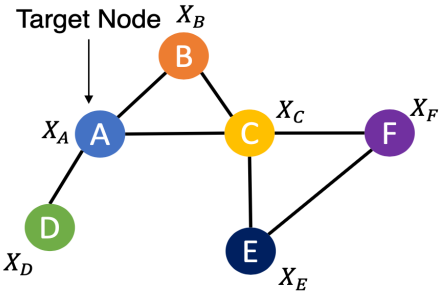


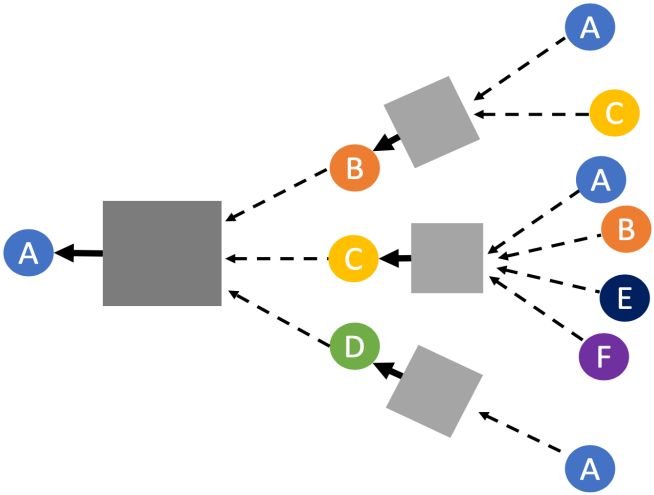


Source: Minji Yoon, CMU

What is GNN? – Forward propagation

-





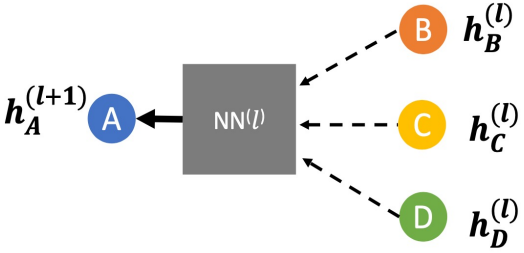
Source: Minji Yoon, CMU

What is GNN? – Forward propagation

1. Aggregate messages from neighbors

$h_v^{(l)}$: node embedding of v at l-th layer
$\mathcal{N}(v)$: neighboring nodes of v
$\boldsymbol{f}^{(l)}$: aggregation function at l-th layer
$m_v^{(l)}$: message vector of v at l-th layer

$$m_A^{(l)} = \boldsymbol{f}^{(l)}\left(h_A^{(l)}, \left\{h_u^{(l)} : u \in \mathcal{N}(A)\right\}\right)$$
$$= \boldsymbol{f}^{(l)}\left(h_A^{(l)}, h_B^{(l)} h_C^{(l)} h_D^{(l)}\right)$$



Neighbors of node A
$\mathcal{N}(A) = \{B, C, D\}$

Source: Minji Yoon, CMU

What is GNN? – Forward propagation

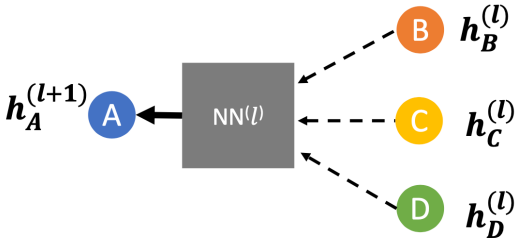
1. Aggregate messages from neighbors

$$m_A^{(l)} = \boldsymbol{f}^{(l)}\left(h_A^{(l)}, \left\{h_u^{(l)} : u \in \mathcal{N}(A)\right\}\right)$$
$$= \boldsymbol{f}^{(l)}\left(h_A^{(l)}, h_B^{(l)} h_C^{(l)} h_D^{(l)}\right)$$

2. Transform messages

$\boldsymbol{g}^{(l)}$: transformation function at l-th layer

$$h_A^{(l+1)} = \boldsymbol{g}^{(l)}(m_A^{(l)})$$



Neighbors of node A

$\mathcal{N}(A) = \{B, C, D\}$

Source: Minji Yoon, CMU

What is GNN? – Forward propagation

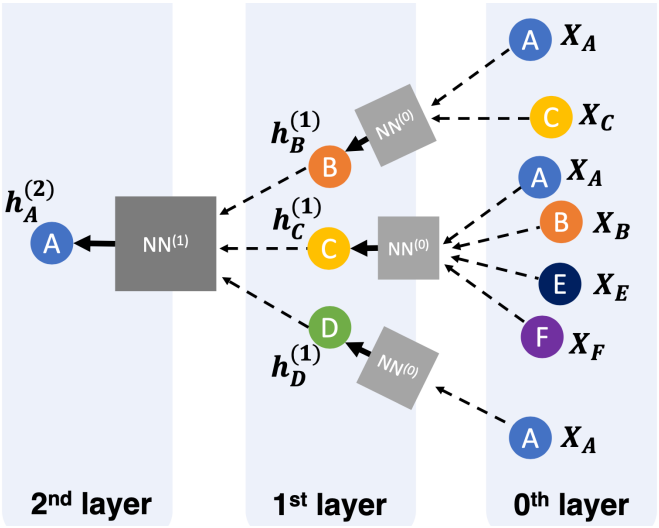
In each layer l ,
for each target node v :

1. Aggregate messages

$$m_v^{(l)} = \boldsymbol{f}^{(l)}\left(h_v^{(l)}, \left\{h_u^{(l)} : u \in \mathcal{N}(v)\right\}\right)$$

2. Transform messages

$$h_v^{(l+1)} = \boldsymbol{g}^{(l)}(m_v^{(l)})$$



Source: Minji Yoon, CMU

What is GNN? – Forward propagation

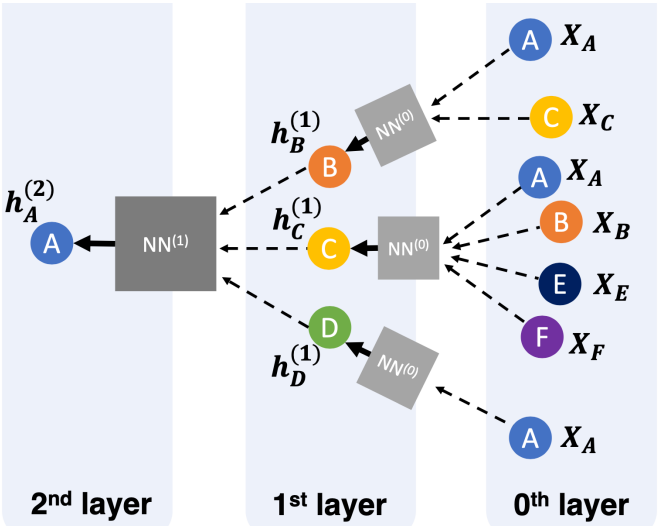
Graph Convolutional Networks[1]

1. Aggregate messages

$$m_v^{(l)} = \frac{1}{|\mathcal{N}(v)+1|} \sum_{u \in \mathcal{N}(v) \cup \{v\}} h_u^{(l)}$$

2. Transform messages

$$h_v^{(l+1)} = \sigma(\boldsymbol{W}^{(l)} \circ m_v^{(l)})$$



[1] Kipf, Thomas N., et al. "Semi-supervised classification with graph convolutional networks."

Source: Minji Yoon, CMU

What is GNN? – Forward propagation

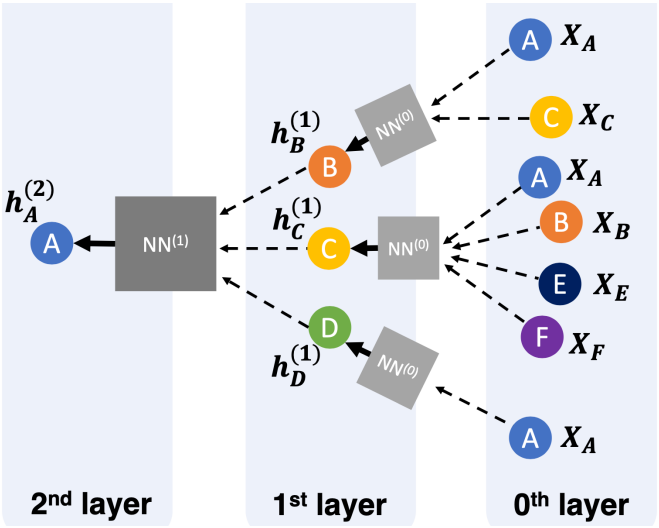
Graph Isomorphism Networks[2]

1. Aggregate messages

$$m_v^{(l)} = \sum_{u \in \mathcal{N}(v) \cup \{v\}} h_u^{(l)}$$

2. Transform messages

$$h_v^{(l+1)} = \sigma(\boldsymbol{W}^{(l)} \circ m_v^{(l)})$$



[2] Xu, Keyulu, et al. "How powerful are graph neural networks?."

Source: Minji Yoon, CMU

What is GNN? – Forward propagation

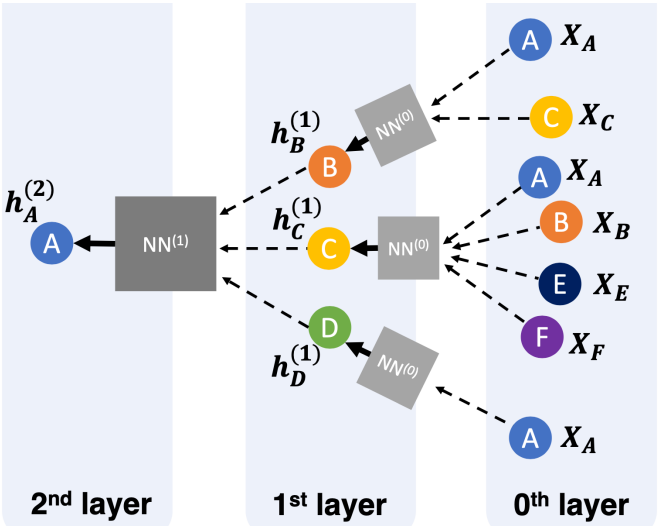
Simplified GCN[3]

1. Aggregate messages

$$m_v^{(l)} = \frac{1}{|\mathcal{N}(v)+1|} \sum_{u \in \mathcal{N}(v) \cup \{v\}} h_u^{(l)}$$

2. Transform messages

$$h_v^{(l+1)} = \boldsymbol{W}^{(l)} \circ m_v^{(l)}$$



[3] Wu, Felix, et al. "Simplifying graph convolutional networks."

Source: Minji Yoon, CMU

What is GNN? – Forward propagation

In each layer l :

Aggregate over neighbors

$$m_v^{(l-1)} = \boldsymbol{f}^{(l)}\left(h_v^{(l-1)}, \left\{h_u^{(l-1)} : u \in \mathcal{N}(v)\right\}\right)$$

Core part of GNNs

Transform messages

$$h_v^{(l)} = \boldsymbol{g}^{(l)}(m_v^{(l-1)})$$

1-layer MLP is commonly used

Source: Minji Yoon, CMU

Graph Convolutional Network (GCN)

- GCN[1]
  - Average embeddings of neighboring nodes

$$m_v^{(l)} = \frac{1}{|\mathcal{N}(v)+1|} \sum_{u \in \mathcal{N}(v) \cup \{v\}} h_u^{(l)}$$

$$h_v^{(l+1)} = \sigma(\boldsymbol{W}^{(l)} \circ m_v^{(l)})$$

+

$$h_v^{(l)} = \sigma(\boldsymbol{W}^{(l)} \circ (\frac{1}{|\mathcal{N}(v)+1|} \sum_{u \in \mathcal{N}(v) \cup \{v\}} h_u^{(l-1)}))$$

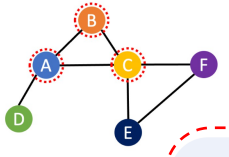
[1] Kipf, Thomas N., et al. "Semi-supervised classification with graph convolutional networks."

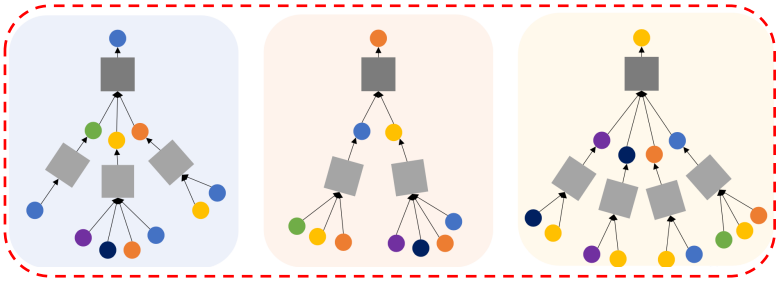
Graph Convolutional Network (GCN)

- GCN[1]

Can we use batch-mode?

$$h_v^{(l)} = \sigma(\boldsymbol{W}^{(l)} \circ (\frac{1}{|\mathcal{N}(v)+1|} \sum_{u \in \mathcal{N}(v) \cup \{v\}} h_u^{(l-1)}))$$





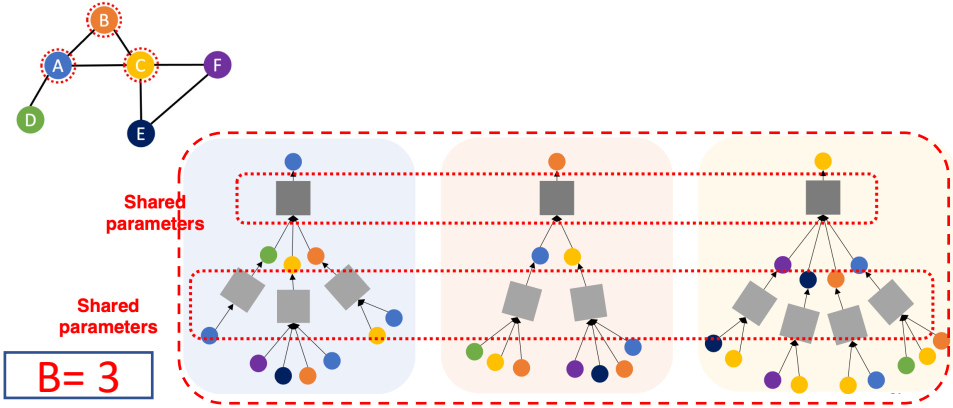
B= 3

[1] Kipf, Thomas N., et al. "Semi-supervised classification with graph convolutional networks."

Graph Convolutional Network (GCN)

- GCN[1]

$$h_v^{(l)} = \sigma(\boldsymbol{W}^{(l)} \circ (\frac{1}{|\mathcal{N}(v)+1|} \sum_{u \in \mathcal{N}(v) \cup \{v\}} h_u^{(l-1)}))$$



[1] Kipf, Thomas N., et al. "Semi-supervised classification with graph convolutional networks."

Graph Convolutional Network (GCN)

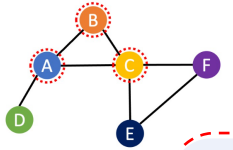
- GCN[1]

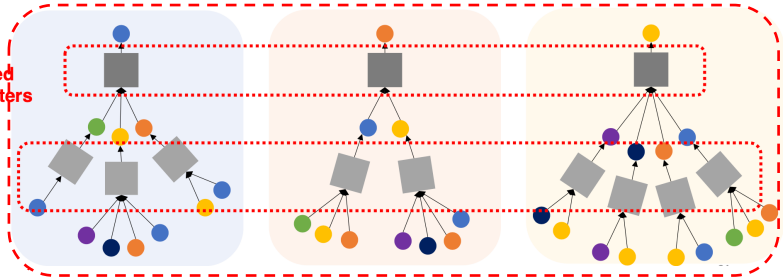
$$h_v^{(l)} = \sigma(\boldsymbol{W}^{(l)} \circ (\frac{1}{|\mathcal{N}(v)+1|} \sum_{u \in \mathcal{N}(v) \cup \{v\}} h_u^{(l-1)}))$$

$$\boldsymbol{H}^{(l)} = \sigma(\widetilde{(\boldsymbol{A}+\boldsymbol{I})}\boldsymbol{H}^{(l-1)}\boldsymbol{W}^{(l)})$$

Node embedding matrix

(row-normalized) **Adjacency matrix**





B= 3

[1] Kipf, Thomas N., et al. "Semi-supervised classification with graph convolutional networks."

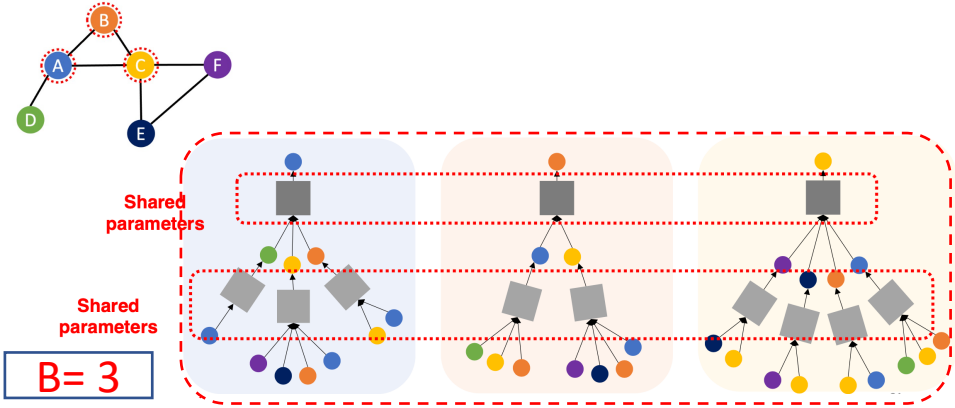
Graph Convolutional Network (GCN)

- GCN[1]

$$h_v^{(l)} = \sigma(\boldsymbol{W}^{(l)} \circ (\frac{1}{|\mathcal{N}(v)+1|} \sum_{u \in \mathcal{N}(v) \cup \{v\}} h_u^{(l-1)}))$$

$$\boldsymbol{H}^{(l)} = \sigma(\widetilde{(\boldsymbol{A}+\boldsymbol{I})}\boldsymbol{H}^{(l-1)}\boldsymbol{W}^{(l)})$$

Fixed **Trainable**



[1] Kipf, Thomas N., et al. "Semi-supervised classification with graph convolutional networks."

Graph Convolutional Network (GCN)

$$h_v^{(l)} = \sigma\left(W^{(l)} \circ \left(\frac{1}{|\mathcal{N}(v)+1|}\sum_{u \in \mathcal{N}(v)\cup\{v\}} h_u^{(l-1)}\right)\right)$$

$$H^{(l)} = \sigma\left((\widetilde{A+I})H^{(l-1)}W^{(l)}\right)$$

Node embedding matrix

(row-normalized) **Adjacency matrix**

Assume A is the affinity matrix

Let D be diagonal matrix where

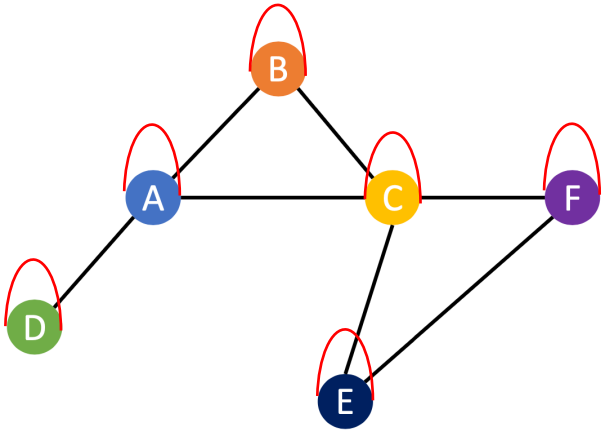
$D_{v,v} = \mathrm{Deg}(v) = |N(v)|$

- The inverse of D: D^{-1} is also diagonal:
  $D_{v,v}^{-1} = 1/|N(v)|$

$$(D+I)^{-1} * (A+I) * H^{(l-1)}$$

Graph Convolutional Network (GCN)

$$\widetilde{A+I} = (D+I)^{-1} * (A+I) \longrightarrow \tilde{A} = D^{-1} * A$$



A graph is composed of

- **Nodes** (also called vertices)
- **Edges** connecting a pair of nodes

presented in an **adjacency matrix**

	A	B	C	D	E	F
A	1	1	1	1		
B	1	1	1			
C	1	1	1		1	1
D	1			1		
E			1		1	1
F			1		1	1

$$\mathcal{L} = \mathbf{I} - (\mathbf{D})^{-1} * (\mathbf{A})$$

Normalized Graph Laplacian

Graph Convolutional Network (GCN) -- Summary

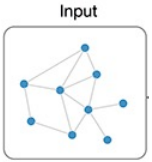
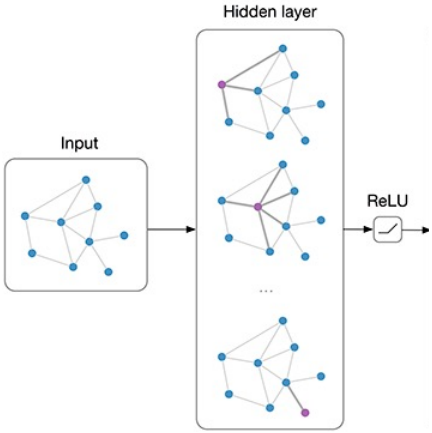


Image Credit: Defferrard et al. NIPS 2016

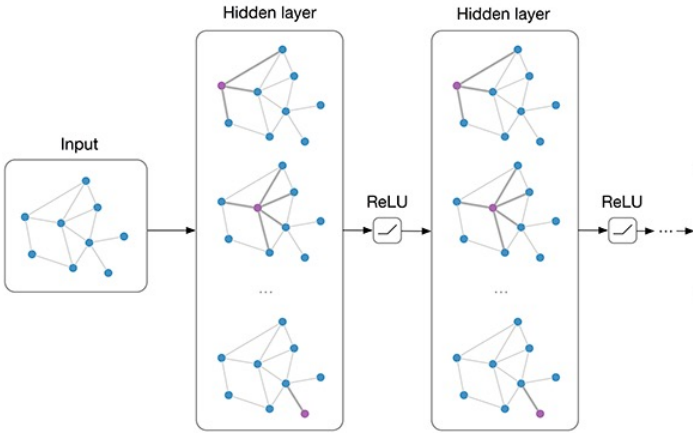
Graph Convolutional Network (GCN) -- Summary



$$\mathrm{ReLU}\left(\hat{A}XW^{(0)}\right)$$

Image Credit: Defferrard et al. NIPS 2016

Graph Convolutional Network (GCN) -- Summary



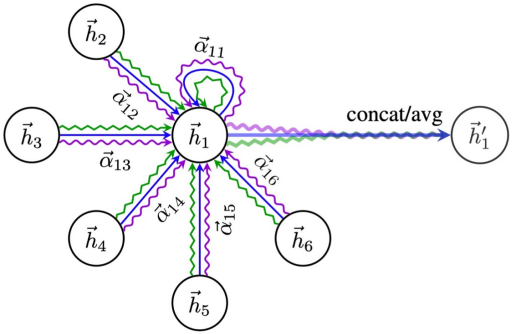
$$\hat{A}\,\mathrm{ReLU}\left(\hat{A}XW^{(0)}\right)W^{(1)}$$

Image Credit: Defferrard et al. NIPS 2016

Graph Attention Network (optional)

- GAT[14]
  - Different weights to different nodes in a neighborhood
  - Multi-head attention

$$\alpha_{ij} = \frac{\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_j]\right)\right)}{\sum_{k\in\mathcal{N}_i}\exp\left(\text{LeakyReLU}\left(\vec{\mathbf{a}}^T[\mathbf{W}\vec{h}_i \| \mathbf{W}\vec{h}_k]\right)\right)}$$



[14] Petar Veličković., et al. "GRAPH ATTENTION NETWORKS."

How to Train GNN?

- Semi-supervised learning
  - Input node features are given for all nodes in a graph
  - Only a subset of nodes have labels

$$\min_{\theta} \mathcal{L}(\boldsymbol{y}, f(\boldsymbol{z}_v))$$

$\boldsymbol{y}$: node label

$\mathcal{L}$ could be L2 if $\boldsymbol{y}$ is real number, or cross entropy if $\boldsymbol{y}$ is categorical

Node embedding $\boldsymbol{z}_v$ is a function of input graph

How to Train GNN?

Unsupervised setting:

- No node label available
- **Use the graph structure as the supervision!**
  - **"Similar" nodes have similar embeddings**

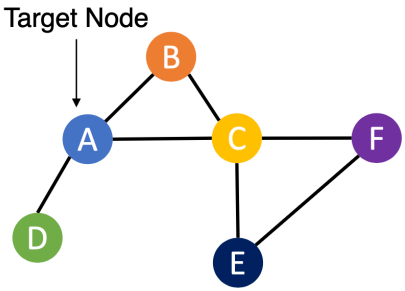
$$\mathcal{L} = \sum_{z_u, z_v} \mathrm{CE}(y_{u,v}, \mathrm{DEC}(z_u, z_v))$$

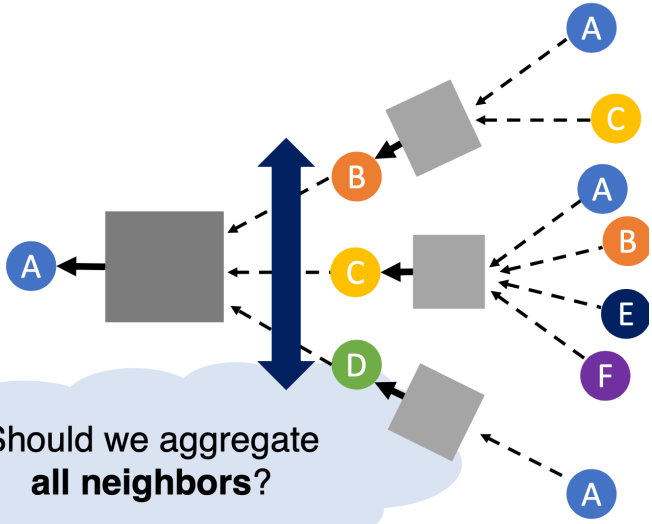
  - Where $y_{u,v} = 1$ when node u and v are **similar**
  - CE is the cross entropy
  - DEC is the decoder such as inner product

Still an active research topic!

Two interesting questions about GNN

Question 1: Width?

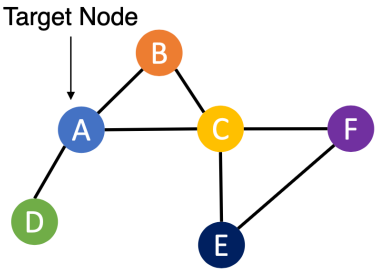


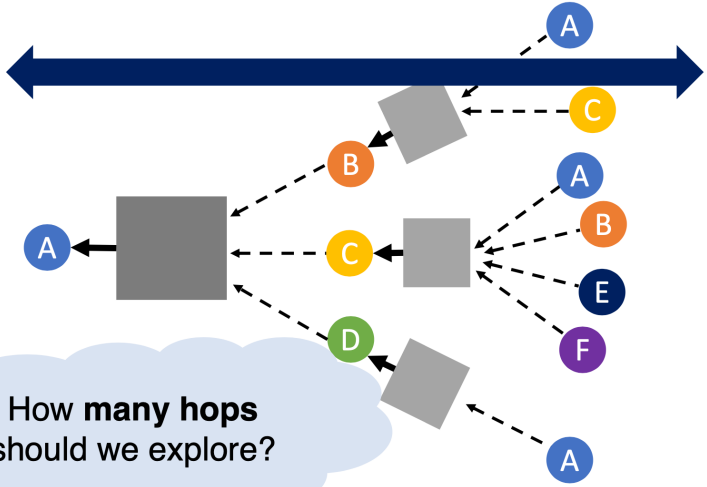


Source: Minji Yoon, CMU

Two interesting questions about GNN

Question 2: Depth?





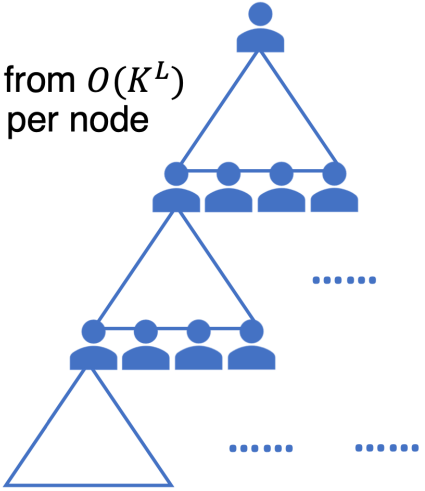
How **many hops** should we explore?

Source: Minji Yoon, CMU

Two interesting questions about GNN

Question 1: Width?

- If we aggregate all neighbors, GNNs have scalability issues
- Neighbor explosion
  - In L -layer GNNs, one node aggregates information from $O(K^L)$ nodes where K is the average number of neighbors per node



Source: Minji Yoon, CMU

Two interesting questions about GNN

Question 1: Width?

- If we aggregate all neighbors, GNNs have scalability issues
- Neighbor explosion
  - Hub nodes who are connected to a huge number of nodes

Cristiano Ronaldo

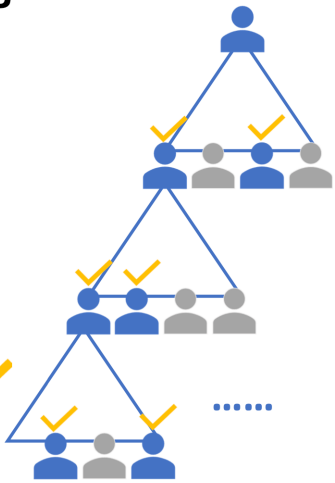
Cristiano Ronaldo is currently the most-followed individual on Facebook, with over 150 million followers.

Source: Minji Yoon, CMU

Two interesting questions about GNN

Question 1: Width?

- Limit the neighborhood expansion by **sampling** a fixed number of neighbors



Source: Minji Yoon, CMU

Two interesting questions about GNN

Question 1: Width?

- Random sampling
  - Assign **same** sampling probabilities to all neighbors
  - *GraphSage*[4]
- Importance sampling
  - Assign **different** sampling probabilities to all neighbors
  - *FastGCN*[5], *LADIES*[6], *AS-GCN*[7], *GCN-BS*[8], *PASS*[9]

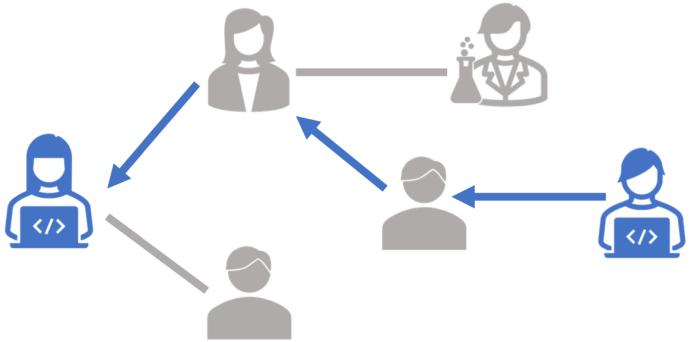
[4] Will Hamilton, et al. "Inductive representation learning on large graphs"
[5] Jie Chen, et al. "Fastgcn: fast learning with graph convolutional networks via importance sampling"
[6] Difan Zou, et al. "Layer-Dependent Importance Sampling for Training Deep and Large Graph Convolutional Networks"
[7] Wenbing Huang, et al. "Adaptive sampling towards fast graph representation learning"
[8] Ziqi Liu, et al. "Bandit Samplers for Training Graph Neural Networks"
[9] Minji Yoon, et al. "Performance-Adaptive Sampling Strategy Towards Fast and Accurate Graph Neural Networks"

Source: Minji Yoon, CMU

Two interesting questions about GNN

Question 2: Depth?

- Informative neighbors could be indirectly connected with a target node



Source: Minji Yoon, CMU

Two interesting questions about GNN

Question 2: Depth?

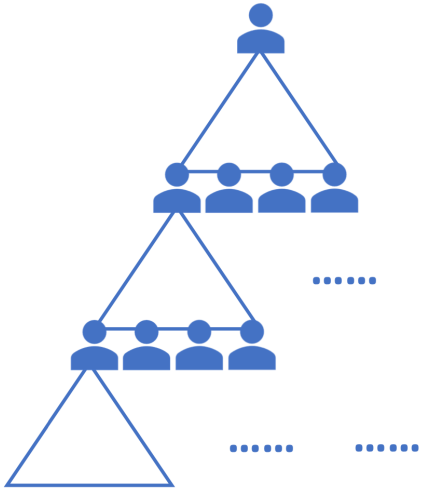
- 2-layer or 3-layer GNNs are commonly used in real worlds

Source: Minji Yoon, CMU

Two interesting questions about GNN

Question 2: Depth?

- When we increase the depth L more than this, GNNs face neighbor explosion $O(K^L)$
  - **Over-smoothing**



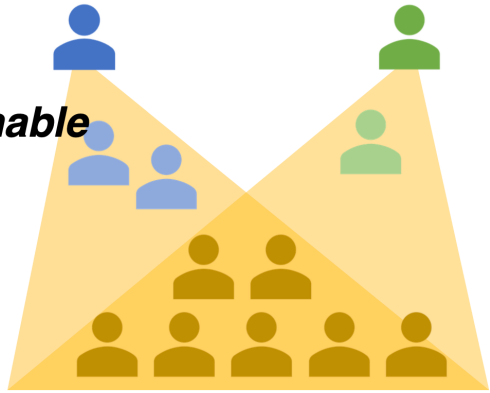
Source: Minji Yoon, CMU

Two interesting questions about GNN

Question 2: Depth?

Over-smoothing[10]

- When GNNs become deep, nodes share many neighbors
- Node embeddings become ***indistinguishable***



[10] Qimai Li, et al. "Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning"

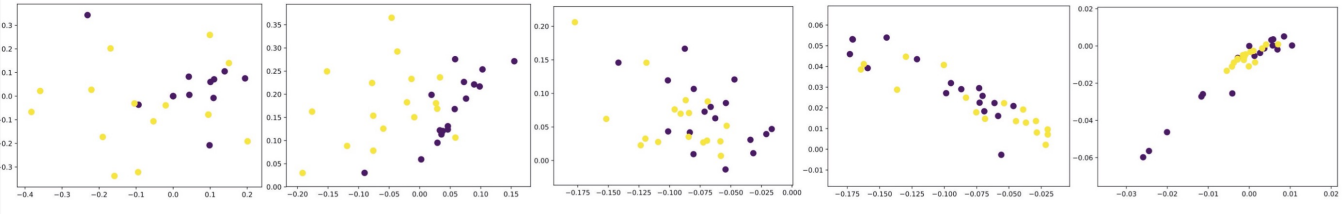
Source: Minji Yoon, CMU

Two interesting questions about GNN

Question 2: Depth?

Over-smoothing[10]

- Node embeddings of Zachary's karate club network with GNNs



(a) 1-layer (b) 2-layer (c) 3-layer (d) 4-layer (e) 5-layer

[10] Qimai Li, et al. "Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning"

Source: Minji Yoon, CMU

GNN Applications

Food Discovery with Uber Eats: Using Graph Learning to Power Recommendations

Ankit Jain, Isaac Liu, Ankur Sarda, and Piero Molino December 4, 2019

0

Pinterest Engineering

Aug 15, 2018 · 8 min read

PinSage: A new graph convolutional neural network for web-scale recommender systems

Ruining He | Pinterest engineer, Pinterest Labs

Web image search gets better with graph neural networks

…h to image search uses images returned by traditional search …dels in a graph neural network through which similarity signals are …hieving improved ranking in cross-modal retrieval.

…ural Network

…ER LABS Europe NAVER LABS FOLLOW

amazon | science

PUBLICATION

P-Companion: A principled framework for diversified complementary product recommendation

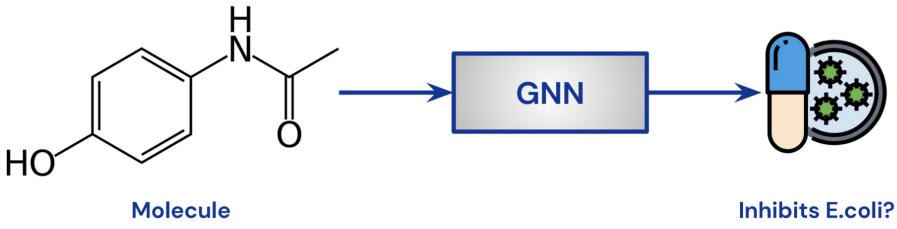
By Junheng Hao, Tong Zhao, Jin Li, Xin Luna Dong, Christos Faloutsos, Yizhou Sun, Wei Wang

2020

Source: Minji Yoon, CMU

GNN applications

- Graph-level prediction: whether the molecule is a potent **drug**[29]
  - Execute on a large dataset of known candidate molecules
  - Select the ~ *top-100* candidates from the GNN model
  - Have chemists thoroughly investigate those

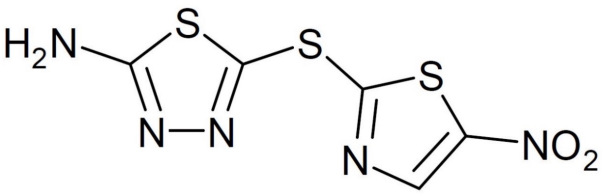


[29] Jonathan M.Stokes, et al. "A Deep Learning Approach to Antibiotic Discovery"

Source: Minji Yoon, CMU

GNN applications

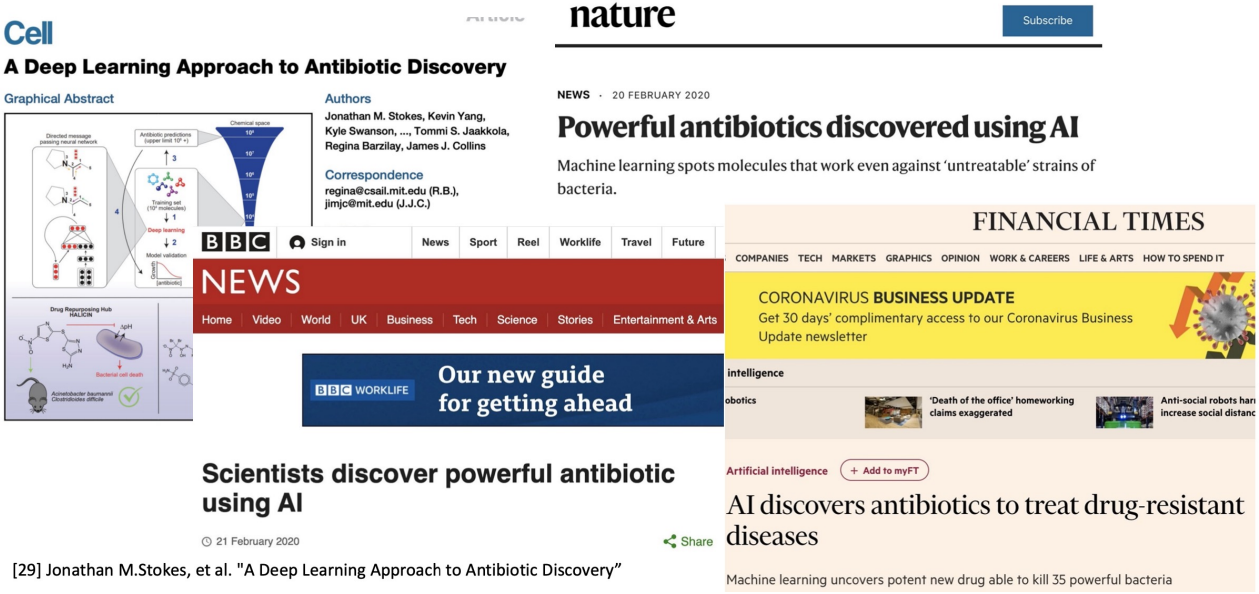
- Discover a previously overlooked compound that is a **highly potent** antibiotic[29]



Halicin

[29] Jonathan M.Stokes, et al. "A Deep Learning Approach to Antibiotic Discovery"

GNN applications



[29] Jonathan M.Stokes, et al. "A Deep Learning Approach to Antibiotic Discovery"

After the break

After the break: **Reinforcement Learning: Policy Gradient**

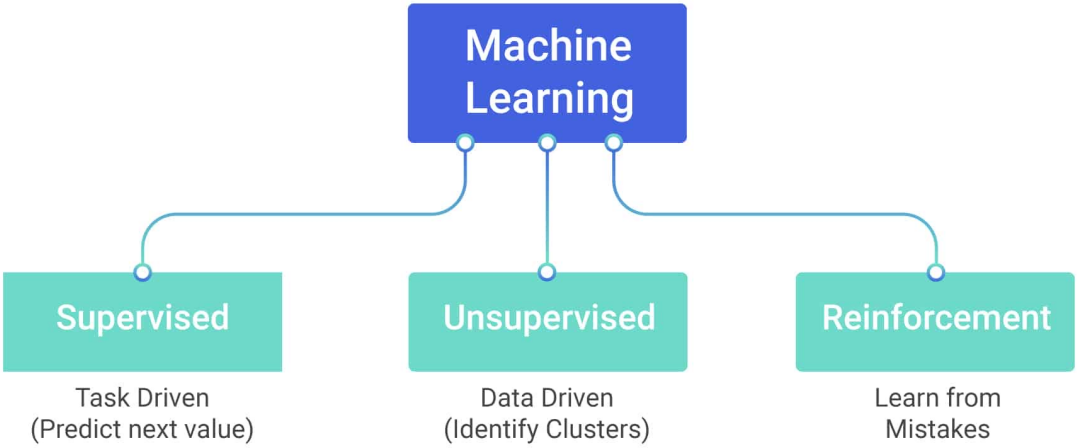
Alpha-Go Trailer --- This is it, folks!

Overview

- Most of this course was about supervised learning, plus a little unsupervised learning.
- Reinforcement learning:
  - Middle ground between supervised and unsupervised learning
  - An agent acts in an environment and receives a reward signal.

- Next lecture: combine policies and Q-learning

Overview



Source: https://perfectial.com/blog/reinforcement-learning-applications/

Overview

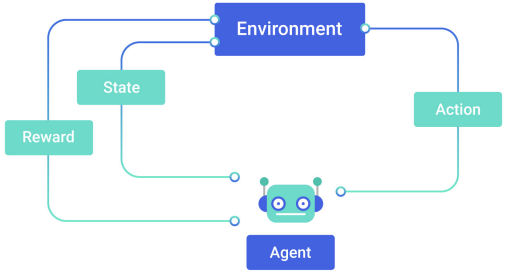
How does AI take over the world? Three Steps!

SUPERVISED LEARNING

UNSUPERVISED LEARNING

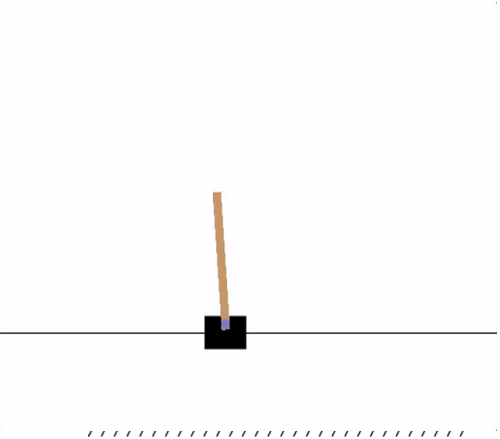
REINFORCEMENT LEARNING

Reinforcement learning



- An **agent** interacts with an **environment** (e.g. game of Breakout)
- In each time step t,
  - the agent receives **observations** (e.g. pixels) which give it information about the **state** $\mathbf{s}_t$ (e.g. positions of the ball and paddle)
  - the agent picks an **action** $\mathbf{a}_t$ (e.g. keystrokes) which affects the state
- The agent periodically receives a **reward** $r(\mathbf{s}_t, \mathbf{a}_t)$, which depends on the state and action (e.g. points)
- The agent wants to learn a **policy** $\pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t)$
  - Distribution over actions depending on the current state and parameters θ

Cart-Pole Problem



Objective: Balance a pole on top of a movable cart

State: angle, angular speed, position, horizontal velocity
Action: horizontal force applied on the cart
Reward: 1 at each time step if the pole is upright

Source: Fei-Fei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Robot Locomotion

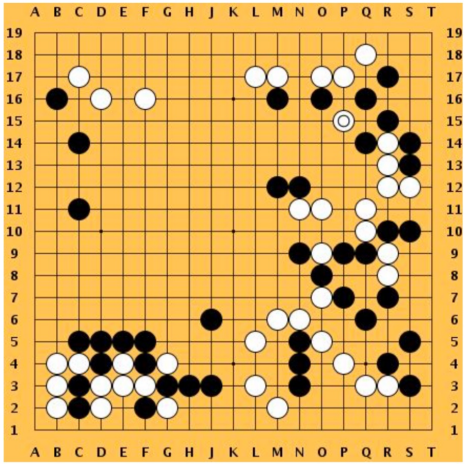
Objective: Make the robot move forward

State: Angle and position of the joints
Action: Torques applied on joints
Reward: 1 at each time step upright + forward movement

Source: Fei-Fei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

Reinforcement learning

Go



Objective: Win the game!

State: Position of all pieces
Action: Where to put the next piece down
Reward: 1 if win at the end of the game, 0 otherwise

Source: Fei-Fei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**

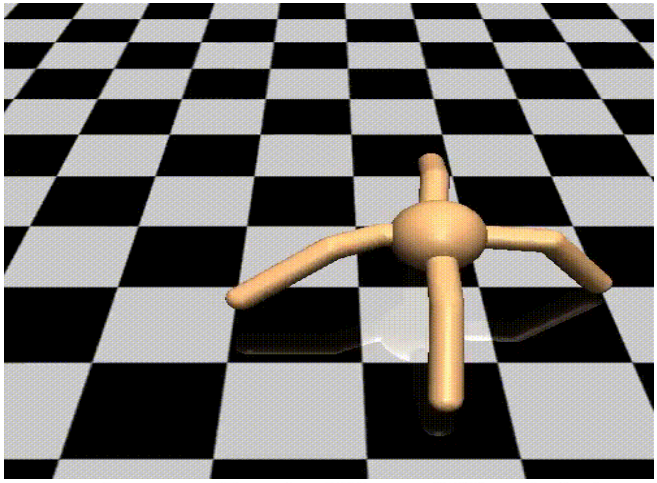
Markov Decision Processes

- The environment is represented as a Markov decision process $\mathcal{M}$.
- Markov assumption: all relevant information is encapsulated in the current state; i.e. the policy, reward, and transitions are all independent of past states given the current state
- Components of an MDP:
  - initial state distribution $p(\mathbf{s}_0)$
  - policy $\pi_\theta(\mathbf{a}_t \,|\, \mathbf{s}_t)$
  - transition distribution $p(\mathbf{s}_{t+1} \,|\, \mathbf{s}_t, \mathbf{a}_t)$
  - reward function $r(\mathbf{s}_t, \mathbf{a}_t)$
- Assume a fully observable environment, i.e. $\mathbf{s}_t$ can be observed directly
- Rollout, or trajectory $\tau = (\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, \ldots, \mathbf{s}_T, \mathbf{a}_T)$
- Probability of a rollout

$$p(\tau) = p(\mathbf{s}_0)\, \pi_\theta(\mathbf{a}_0 \,|\, \mathbf{s}_0)\, p(\mathbf{s}_1 \,|\, \mathbf{s}_0, \mathbf{a}_0) \cdots p(\mathbf{s}_T \,|\, \mathbf{s}_{T-1}, \mathbf{a}_{T-1})\, \pi_\theta(\mathbf{a}_T \,|\, \mathbf{s}_T)$$

Markov Decision Processes

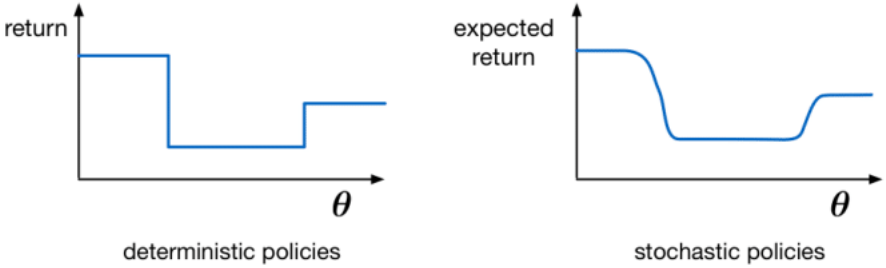
Continuous control in simulation, e.g. teaching an ant to walk

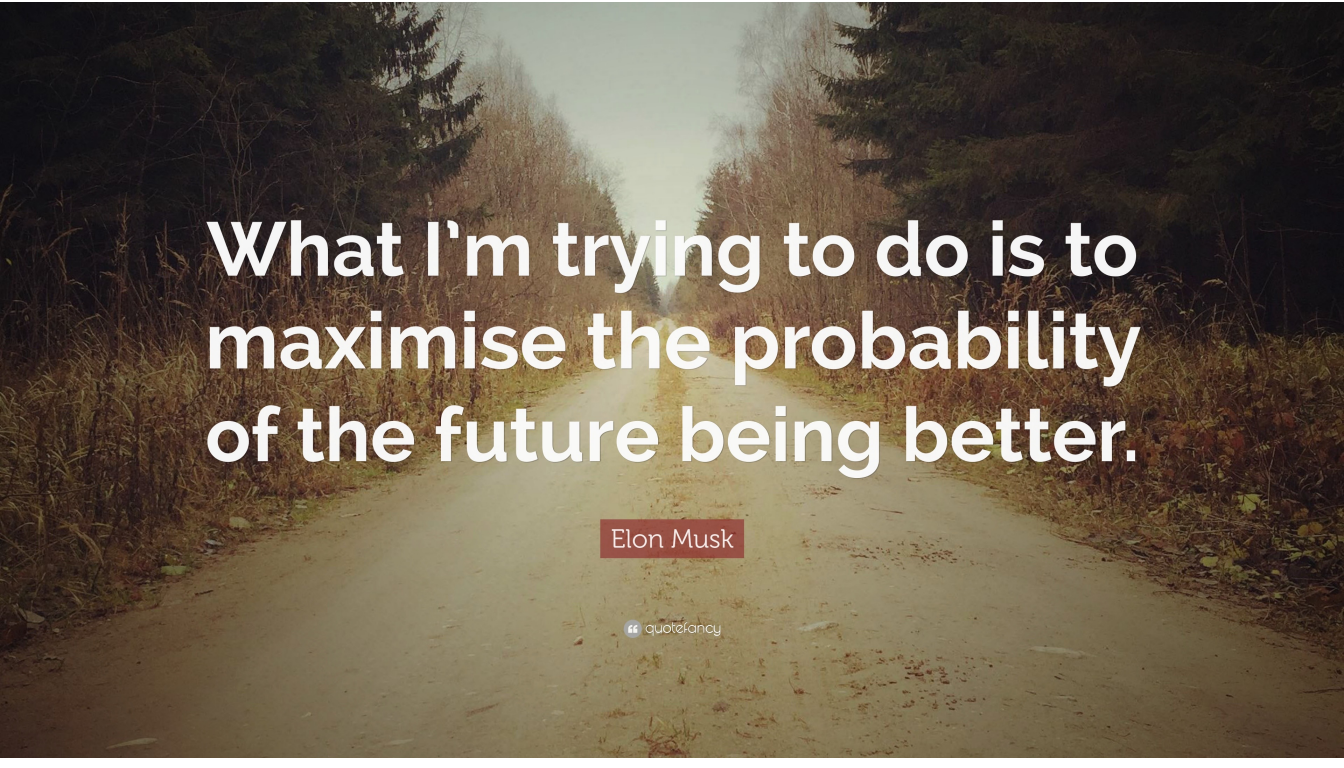


- State: positions, angles, and velocities of the joints
- Actions: apply forces to the joints
- Reward: distance from starting point
- Policy: output of an ordinary MLP, using the state as input
- More environments: `https://gym.openai.com/envs/#mujoco`

Markov Decision Processes

- Return for a rollout: $r(\tau) = \sum_{t=0}^{T} r(\mathbf{s}_t, \mathbf{a}_t)$
  - Note: we're considering a finite horizon T, or number of time steps; we'll consider the infinite horizon case later.
- Goal: maximize the expected return, $R = \mathbb{E}_{p(\tau)}[r(\tau)]$
- The expectation is over both the environment's dynamics and the policy, but we only have control over the policy.
- The stochastic policy is important, since it makes R a continuous function of the policy parameters.
  - Reward functions are often discontinuous, as are the dynamics (e.g. collisions)




What I'm trying to do is to maximise the probability of the future being better.
Elon Musk
quotefancy