ECE598: Generative AI Models

March 9, 2022

1 Lecture 2-3: Normalizing Flows

Learning objectives

- Derive and use the probability integral transformation, including the Box-Muller transform
- Perform transformations of pdfs under linear mappings
- Perform transformations of pdfs under nonlinear mappings
- Describe alternatives to the probability integral transformation for generating samples, e.g. rejection sampling and MCMC methods
- Discuss the unseen elements problem, Laplacian smoothing, and the Good-Turing estimator
- Mention the main idea of normalizing flows
- Discuss relationships between generators and density estimators in context of implicit/explicit representations.
- Use techniques of transforming pdfs under nonlinear mappings to derive the main equations of normalizing flows
- Use LOTUS property in context of normalizing flows
- Describe evolution of densities in infinitesimal flows and in discrete flows
- Draw and describe the GLOW architecture
- Specify applications of normalizing flows, e.g. for generating datasets or finding Bayes optimal limits

1.1 Generating a random variable with specified distribution

1.1.1 Probability integral transformation

- Want to generate sample from random variable (RV) with given cumulative distribution function (CDF)
- Have access to realization of $U(0,1)$

① Apply F_x to X
Let $Y = F_x(x)$. Since F_x is non-decreasing from 0 to 1, there is a value C_v such that $F_x(C_v) = V$
$\Rightarrow Pr[F_x(x) \leq V] = Pr[x \leq C_v] = F_x(C_v) = V$
So $F_x(x)$ is $U(0,1)$.

② Go backwards
Apply $F_x^{-1}(.)$ to uniform RV $U(0,1)$, call u
Should replace rv with cdf F_x
$F^{-1}(u) = min\{c : F(c) \geq u\}$
For real C_0 and U_0, with $0 < U_0 < 1$
$F^{-1}(U_0) \leq C_0$, iff $U_0 \leq F(C_0)$.
If $X = F^{-1}(V)$, then $F_x(c) = Pr[F^{-1}(V) \leq C] = Pr[V \leq F(c)] = F(c)$
In ML, $g(.) = F^{-1}(.)$ is called a generator. In Statistics, it's called simulation.

$U(0,1) \xrightarrow{g} N(0,1)$, **Box-Muller transform**
Suppose u_1, u_2 are $iid \sim U(0,1)$, Let
$Z_0 = \sqrt{-2ln\ u_1}cos(2\pi u_2) = Rcos(\Theta)$
$Z_1 = \sqrt{-1ln\ u_1}sin(2\pi u_2) = Rsin(\Theta)$, where $R^2 = -2ln\ u_1, \Theta = 2\pi u_2$ in polar coordinates.
Then $Z_1 \perp Z_2$ (independent), both $\sim N(0,1)$

For discrete distributions
Gambel-Max generator:
If $Z \sim U(0,1)$, then $g(Z) \sim P$
If $g(Z) = \underset{x}{argmax}(log\ P(x) - log\ log(1/Z))$

1.1.2 Transformations of pdfs under linear mappings

Suppose we have joint pdf $f_{xy}(u,v)$ of $RV(x,y)$, vector form $f_{xy}(\begin{bmatrix} u \\ v \end{bmatrix})$
Have 2 new rv (W,Z), linear function of (X,Y), find $f_{W,Z}$
eg. $\begin{bmatrix} W \\ Z \end{bmatrix} = A\begin{bmatrix} X, Y \end{bmatrix}$, where $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, $det(A) = ad - bc$.
If $det(A) \neq 0$, A has inverse $A^{-1} = \frac{1}{det(A)}\begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$

Theorem : Suppose $\begin{bmatrix} W \\ Z \end{bmatrix} = A\begin{bmatrix} X \\ y \end{bmatrix}$, where $\begin{bmatrix} X \\ Y \end{bmatrix} \sim f_{xy}$, $det(A) \neq 0$, then $\begin{bmatrix} W \\ Z \end{bmatrix}$ has joint pdf $f_{W,Z}(\begin{bmatrix} \alpha \\ \beta \end{bmatrix}) = \frac{1}{|det(A)|}f_{xy}(A^{-1}(\begin{bmatrix} \alpha \\ \beta \end{bmatrix}))$

1.1.3 Transformations of pdfs under nonlinear mappings

$\begin{bmatrix} W \\ Z \end{bmatrix} = g(\begin{bmatrix} X \\ Y \end{bmatrix})$
Recall Jacobian of g,
$J = J(u,v) = \begin{bmatrix} \frac{\partial\ g_1(u,v)}{\partial\ u} & \frac{\partial\ g_1(u,v)}{\partial\ v} \\ \frac{\partial\ g_2(u,v)}{\partial\ u} & \frac{\partial\ g_2(u,v)}{\partial\ v} \end{bmatrix}$, where g_1, g_2 are coordinates of g.

Theorem : Suppose $\begin{bmatrix} W \\ Z \end{bmatrix} = g(\begin{bmatrix} X \\ Y \end{bmatrix})$, where $\begin{bmatrix} X \\ Y \end{bmatrix}$ has pdf f_{xy} and g is a one-to-one mapping from support of f_{xy} to $\mathbb{R}^2$.
Suppose J of g exists, is continuous and has a non-zero determinant everywhere
$f_{WZ}(\begin{bmatrix} \alpha \\ \beta \end{bmatrix}) = \frac{1}{|det(J)|}f_{xy}(g^{-1}\begin{bmatrix} \alpha \\ \beta \end{bmatrix})$ for (α, β) in support of f_{WZ}. If g is many to one, just sum over pieces.

1.1.4 Alternative approach to sample from distributions

:

- rejection sampling
- Markov chain Monte Carlo (MCMC)

1.1.5 Unseen elements problem

If we just have iid samples from P, estimate $\hat{P}$ from P using samples, then generate using $\hat{P}$.
To estimate arbitrary pmf from samples, just use empirical counts. This is actually the maximum likelihood estimate (MLE)
$\hat{P} \to P, \#samples \to \infty$
Techniques for dealing with unseen species (no likelihood):
1. Good-Turing frequency estimating
2. Laplacian smoothing $\Rightarrow$ assign small probability to all possible outcomes.

1.2 Normalizing Flows

1.2.1 Idea of normalizing flows

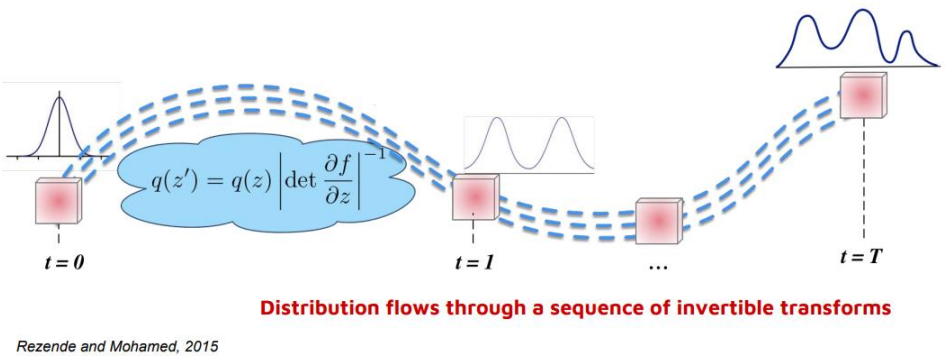


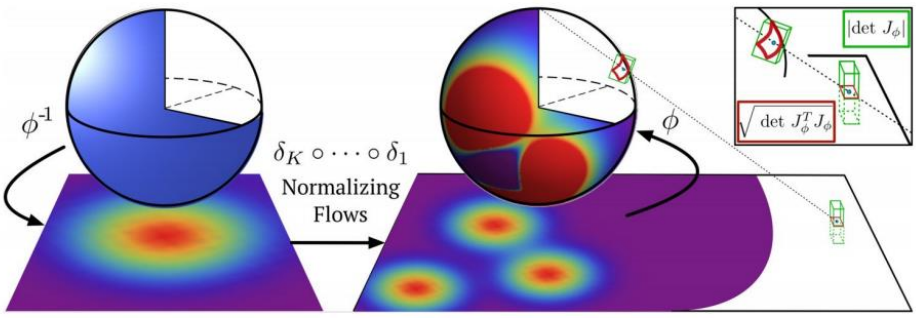
Figure 1: Normalizing Flows. See: https://www.shakirm.com/slides/DeepGenModelsTutorial.pdf

Construct a normalizing flow $\Rightarrow$ a sequence of invertible transformations that convert simple distribution to complex one.

1.2.2 Derivation of the main equations of normalizing flows

Invertible smooth mapping: $f : \mathbb{R}^d \to \mathbb{R}^d$ with $f^{-1} = g$
$g \cdot f(x) = X$ if we use this to transform rv X with distribution q. $\tilde{X} = f(x)$ have distribution.
(*) $q(\tilde{x}) = q(x)|det\frac{\partial\ f^{-1}}{\partial\ x}| = q(x)|det\frac{\partial\ f}{\partial\ x}|^{-1}$
Successively apply (*) when each map is simple.
$X_k = f_k \circ f_{k-1}\circ, ..., \circ f_2 \circ f_1(X_0)$
So $ln(q_k(x_k)) = lnq_0(x_0) - \sum_{k=1}^{K} ln|det\frac{\partial\ f_k}{\partial\ x_{k-1}}|$

- The path traversed by rv $X_k = f_k(x_{k-1})$ is flow. Path successive distribution q_k is normalizing flow. Want inverse and Jacobian to be simple but don't want to lose expressibility.
- Use sequence of invertible transformations until desired complexity.
- Equality is by applying chain rule.



$$\log q_K(\mathbf{z}_K) = \log q_0(\mathbf{z}_0) - \frac{1}{2}\sum_{k=1}^{K} \log\det \left|\mathbf{J}_\phi{}^\top \mathbf{J}_\phi\right|$$

Gemici et al., 2016

Figure 2: https://www.shakirm.com/slides/DeepGenModelsTutorial.pdf

1.2.3 Relationships between generators and density estimators

Continuous setting
→ Parametric density estimation, eg. mean/variance of Gaussian.
→ Non-parametric density estimation, eg. kernel density estimation.
Want to sample from $\hat{P}$, not evaluate $\hat{P}(X)$ at various values. In generative models, we use $\hat{P}$ implicitly, and it's difficult ot backing out $\hat{P}(X)$.
Generative models from learning from data iid samples from P. Try to get estimate $\hat{P}$, then use it to generate new sample, rather than evaluate $\hat{P}(x)$. Often have generative model where $\hat{P}$ is implicit rather than explicit. Backing out $\hat{P}(x)$ is hard.
Suppose $\hat{P}$ is defined implicitly as probability integral transform of density q by $g : \xi \to X$, so for event A:
$Pr[A] = Pr[g^{-1}[A]] = \int_{g^{-1}(A)} q(z)dz = \int_A q(g^{-1}(X))|\nabla_X g^{-1}(X)|dX$,
where $\hat{P}(X) = q(g^{-1}(X))|\nabla_X g^{-1}(X)|$ and when inverse and Jacobian are easy to compute, converting generator to density estimator easily.

1.2.4 LOTUS properties under normalizing flow

LOTUS property: Law of unconscious statistician.
$\mathbb{E}[g(x)] = \sum_k g(x_k)P_x(x_k)$
The expectation w.r.t. transformed density q_k can be computed without explicitly knowing q_k. Any $\mathbb{E}_{q_k}(h(x))$ can be written under q_k without the need to compute the largest Jacobian term after $h(x)$.
$\mathbb{E}_{q_k}[h(x)] = \mathbb{E}_{q_0}[h(f_k \circ f_{k-1 \circ \ldots \circ f_1(x_0)})]$ when $h(x)$ doesn't depend on q_k.

1.2.5 Density in discrete flow

In discrete setting, change of variable formula is easier.
Let x be discrete rv, $Y = f(x)$. The induced pmf of g is sum over pre-image of f:
$P[Y = y] = \sum_{x:f^{-1}(y)} Pr[X = x]$ where $f^{-1}(y)$ is set of all elements to $f(x) = y$.
f is invertible, then $P[Y = y] = P(X = f^{-1}(y))$

1.2.6 Densities in infinitesimal flows

Let the number of transformation $k \to \infty$. Describe how the initial density $q_0(x)$ evolves over time using PDE:
$\frac{\partial}{\partial t} q_t(x) = J_t[q_t(x)]$, J_t is continuous time dynamics.

1.2.7 GLOW

Let X be high-dimensional rv with unknown time distribution $X \sim P(X)$. Collect iid dataset D of size M and model class $P_\theta(X)$ with parameter θ. Consider maximum likelihood, so minimize objective $\mathcal{L}(D) = \frac{1}{N}\sum_{i=1}^{N} - logP_\theta(X^{(i)})$.

For continuous data, $\mathcal{L}(D) \approx \frac{1}{N}\sum_{i=1}^{N} - logP_\theta(\bar{x}^i) + C$, where $\bar{x}^{(i)} = x^{(i)} + u$ with $u \sim U(0, a)$, $C = Mlog\ a$, where a is the quantization level. M is dimension. Optimize using eg. stochastic gradient descent.
In flow-based generation, $Z \sim P_\theta(Z)$ where Z is latent variable and $P_\theta(Z)$ is simple density.
$X = g_\theta(Z)$ where $g_\theta(.)$ is invertible so $Z = f_\theta(x) = g_\theta^{-1}(x)$.
$f = f_1 \circ f_2 \circ ... \circ f_k$
$H_0 = x \overset{f_1}{\longleftrightarrow} H_1 \overset{f_2}{\longleftrightarrow} H2... \overset{f_k}{\longleftrightarrow} Z = H_k$

$$logP_\theta(x) = logP_\theta(Z) + \sum_{k=1}^{K} log|det(\frac{\partial\ H_k}{\partial\ H_{k-1}})|.$$

Easy to compute if Jacobian matrix $det(\frac{\partial\ H_k}{\partial\ H_{k-1}})$ is lower triangular.

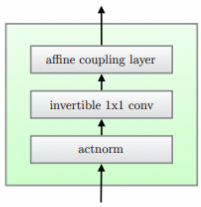
$$log|det(\frac{\partial\ H_k}{\partial\ H_{k-1}})| = \sum(log|diag(\frac{\partial\ H_k}{\partial\ H_{k-1}})|)$$

GLOW architecture

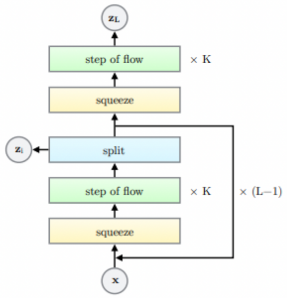
affine coupling layer basically learns a nonlinear independent components representation with easy inverse and easy log det Jacobian, since disentangling is a good representation

invertible convolution is generalization of learned rotation

activation normalization (actnorm) learns affine transformation of activations using a scale and bias parameter per channel



(a) One step of our flow.



(b) Multi-scale architecture (Dinh et al., 2016).

2 Lecture 4-6: Autoencoder

Learning objectives:

- Discuss the basic formulations and structures of autoencoders for applications in dimensionality reduction
- Mathematically describe why undercompleteness, contraction, and denoising might be useful ways of structuring latent spaces
- Specify the deep latent variable model in the language of graphical models
- Describe how graphical models can be parameterized using neural networks
- Introduce the variational formulation of deep latent variable models
- Derive ELBO (variational lower bound)

- Discuss how KL divergence term in ELBO determines two "distances"
- Derive reparameterization trick
- State and manipulate the InfoVAE objective
- Discuss some of the shortcomings of the basic ELBO-based VAE formulation
- Develop the basics of quantization theory
- Describe the VQ-VAE approach

2.1 Basic formulations and structures of autoencoders

An autoencoder is a neural network trained to attempt to copy its input to its output. Internally, it has hidden layer h that describes a code or latent space used to represent the input.
$\underset{input}{\mathbb{X}} \xrightarrow[encoder]{f} \underset{code}{\mathbb{H}} \xrightarrow[decoder]{g} \underset{reconstruction}{\mathbb{R}}$
Tries to set $g(f(x)) = x$ but are designed to be unable to learn to copy perfectly.

- restricted to prevent exact copying
- model forced to prioritize which aspects of the input should be copied, so often learns most useful properties of data

Not just deterministic functions f, g, but also stochastic mappings $P_{encoder}(h|x)$ and $P_{decoder}(x|h)$.

2.2 Structuring latent spaces

2.2.1 Undercompleteness

Constrain h to have smaller dimension than x, to force to have preserved most salient properties. Reminiscent of data compression and very much a form of approximation theory.
Learning process is minimizing loss function $L(x, g(f(x)))$ where L is loss function penalizing $g(f(x))$ for lack of fidelity with x.
When $g(\cdot)$ fixed to be linear and L is mean-square error, an undercomplete autoencoder recovers principle components analysis (PCA), the Karhunen-Loeve transform.
Why? When allowing nonlinear (f,g), get generalization of PCA.
One can also generalize, e.g. with sparsity autoencoder.
$L(x, g(f(x))) + \Omega(h)$, where $\Omega(\cdot)$ is sparsity penalty on latent space h in addition to data fidelity term. Hopefully, this forces learning of relevant things.

2.2.2 Denoising

Minimize $L(x, g(f(\hat{x})))$ in stead of $L(x, g(f(x)))$, where $\hat{x}$ is noisy version of x.
DAE have to undo the noise corruption rather than simply copying, so again try to get most informative elements.

2.2.3 Contraction

Different kind of regularization, to force a function that doesn't change much when x changes a little.
$L(x, g(f(x))) + \Omega(h, x)$ where $\Omega(h, x) = \lambda \sum_i \|\nabla_x h_i\|^2$
or $\Omega(h) = \lambda \| \frac{\partial f(x)}{\partial x} \|_F^2$ where penalty is squared Frobenius norm (sum of squared elements) of Jacobian matrix of partial derivatives associated with encoder function.

2.3 Deep latent variable model under graphical model

AE used for dimensionality reduction, etc, but what about generation?
ⓧ $\overset{P_{encoder}(h|x)}{\longrightarrow}$ ⓗ $\overset{P_{decoder}(x|h)}{\longrightarrow}$ ⓡ
Stochastic encoder/decoder. Note that $P_{encoder}$, $P_{decoder}$ don't have to correspond to same valid joint distribution. $P_{model}(x, h)$ in fact usually don't.

Back up to graphical models before getting to variational autoencoders (VAEs).
Consider conditional model $P_\theta(y|x)$ that approximates underlying conditional distribution $P^*(y|x)$, a distribution over variable y conditioned on input x.
Want to learn $P_\theta(y|x) \approx P^*(y|x)$
→ Graphical models have an interesting calculus, e.g. Forney style graphs ⇔ block diagrams.

2.4 Parameterize graphical model with neural networks

Differentiable feedforward neural networks are a flexible computationally-scalable kind of function approximator (universal approximation theorem).
Learning models with neural networks with many hidden layers is deep learning. Notably NNs can be used to approximate pdfs and pmfs.
→ probability models based on neural networks are computationally scalable since they allow stochastic gradient-based optimizer and scaling to large models, large datasets.
→ think of them as an operator $NN(\blacksquare)$
e.g. neural net can parameterize categorical distribution $P_\theta(y|x)$ over a class label y, conditioned on image x as $P = NN(x), P_\theta(y|x) = categorical(y; P)$

2.5 Variational formulation of deep latent variable models

Consider directed graphical models that have latent variables → latent variables are part of model but not observed directly and not part of dataset denote by Z.
→ Joint distribution $P_\theta(x, Z)$ considers observed variables and latent variables Z.
A deep latent variable model (DLVM) denotes a latent variable model $P_\theta(x, Z)$ whose distributional properties parameterized by neural networks. Also conditional $P_\theta(x, Z|y)$.
Even when each factor (prior or conditional distribution) is relatively simple, marginal $P_\theta(x)$ can be very complex.
A main difficulty of maximum likelihood learning in DLVMs is that marginal probability of data under model is typically intractable since $P_\theta(x) = \int P_\theta(x, Z)dZ$ may not have analytical solution or efficient estimator. Due to intractability, we cannot differentiate w.r.t its parameters and optimize, as we can with fully observed models (Main difficulty is posterior $P_\theta(Z|x)$). There are approximate inference techniques but often computationally intense.
The framework of VAE provides a computationally efficient way of optimizing DLVMs jointly with corresponding inference model using stochastic gradient descent (SGD).
To turn DLVM's intractable learning problem into tractable problem, introduce a parametric inference model $q_\phi(Z|x)$ which is an encoder (recognition model). ϕ are parameters of inference model, called variational parameters. Optimize variance parameters ϕ such that $q_\phi(Z|x) \approx P_\theta(Z|x)$

2.6 ELBO (variational lower bound)

Encoder has a directed graphical model, can be factorized:
$q_\phi = q_\phi(Z_1, ..., Z_n|x) = \prod_{i=1}^{M} q_\phi(Z_i|P_\theta(Z_i), x)$
where $P_\theta(Z_i)$ is set of parent of Z_i in directed graph. Once we have factorization $q_\phi(Z|x)$ can be parameterized using NN where ϕ is weights, bias of NN.

The optimization objective of VAE is evidence lower bound (ELBO). For any choice of $q_\phi(Z|x)$ including choice of ϕ

$$
\begin{aligned}
logP_\theta(x) &= \mathbb{E}_{q_\phi(Z|x)}[logP_\theta(x)] \\
&= \mathbb{E}_{q_\phi(Z|x)}[log\frac{P_\theta(x,Z)}{P_\theta(Z|x)}] \\
&= \mathbb{E}_{q_\phi(Z|x)}[log\frac{P_\theta(x,Z)q_\phi(Z|x)}{P_\theta(Z|x)q_\phi(Z|x)}] \\
&= \mathbb{E}_{q_\phi(Z|x)}[log\frac{P_\theta(x,Z)}{q_\phi(Z|x)}] + \mathbb{E}_{q_\phi(Z|x)}[log\frac{q_\phi(Z|x)}{P_\theta(Z|x)}] \\
&= \mathcal{L}_{\theta,\phi}(x) + D_{KL}(q_\phi(Z|x)||P_\theta(Z|x)) \\
&\Rightarrow [ELBO]
\end{aligned}
\tag{1}
$$

where D_{KL} is non-negative, so
$\mathcal{L}_{\theta,\phi} = \mathbb{E}_{q_\phi(Z|x)}[logP_\theta(x,Z) - log(q_\phi(Z|x))] = logP_\theta(x) - D_{KL}(q_\phi(Z|x)||P_\theta(Z|x))$
$\mathcal{L}_{\theta,\phi} \leq logP_\theta(x) \Rightarrow$ lower bound of log-likelihood.

2.7 KL divergence in ELBO

KL divergence two solutions:

- divergence of approximate posterior from true posterior (???).
- gap between ELBO and $logP_\theta(x)$

Better approximation $\rightarrow$ higher bound.

Maximizing ELBO will optimize:
1. approximately maximize $P_\theta(x)$, so generative model is better
2. minimize KL divergence of approximation $q_\phi(Z|x)$ from $P_\theta(Z|x)$ so $q_\phi(Z|x)$ become better.
Jointly optimize ϕ and θ using SGD

2.8 Reparameterization trick

For continuous latent variables and a differentiable en/decoder, ELBO can be differentiated w.r.t both ϕ and θ if we reparameterize.
First, express rv $Z \sim q_\phi(Z|x)$ as differentiable/invertible transformation of another rv ε.
Given ϕ, x, $Z = g(Z, \phi, x)$, ε independent of x and ϕ.
$\mathbb{E}_{q_\phi(Z|x)}[f(Z)] = \nabla_\phi \mathbb{E}_{p(\varepsilon)}[f(Z)]$
And since expectation and gradient commute by linearity:
$\nabla_\phi \mathbb{E}_{q_\phi(Z|x)}[f(Z)] = \nabla \mathbb{E}_{p(\varepsilon)}[f(Z)] = \mathbb{E}_{p(\varepsilon)}[\nabla_\phi f(Z)] \approx \nabla_\phi f(Z)$
where we can estimate by Monte Carlo:
$\mathcal{L}_{\phi,\theta}(x) = \mathbb{E}_{q_\phi(Z|x)}[log\ P_\theta(x,Z) - log\ q_\phi(Z|x)] = \mathbb{E}_{p(\varepsilon)}[log\ P_\theta(x,Z) - log\ q_\phi(Z|x)]$, where $Z = g(\varepsilon, \phi, x)$.
So we can get single Monte Carlo estimate $\hat{\mathcal{L}}_\theta P(x)$ of individual ELBO.

2.9 Shortcomings of the basic ELBO-based VAE formulation

Approximate inference distribution is often different from true posterior (from ELBO object).

2.10 InfoVAE objective

Can modify ELBO objective itself to balance correct inference and fitting training data?
Assign some weight to balance the two parts? Introduce a mutual information term: $I_q(x, Z)$
$\mathcal{L}_{InfoVAE} = -\lambda D_{KL}(q_\phi(Z)||P_\theta(Z)) - \mathbb{E}_{q(Z)}[D_{KL}(q_\phi(x|Z))||P_\theta(x|Z)] + \alpha I_q(x, Z)$
$= \mathbb{E}_{q_\phi(x,Z)}[logP_\theta(x|Z) - log\frac{q_\phi(Z)^{\lambda+\alpha-1}P_\theta(x)}{P_\theta(Z)^\lambda q_\phi(Z|x)^{\alpha-1}}]$
$= \mathbb{E}_{PD(x)}\mathbb{E}_{q_\phi(Z|x)}[logP_\theta(x|Z)] - (1-\alpha)\mathbb{E}_{PD(x)}D_{KL}(q_\phi(Z|x)||p(Z)) - (\alpha+\lambda-1)D_{KL}(q_\phi(Z)||p(Z))$

VAE map input to a distribution rather vector.
P_θ with parameters θ, relationship between the input x and latent encoding Z.
Prior $P_\theta(Z)$, likelihood $P_\theta(x|Z)$, posterior $P_\theta(Z|x)$.
If we know actual θ^* to generate:
1. a sample $Z^{(c)} \sim P_{\theta^*}(Z)$
2. generate value $x^{(c)}$, $P_{\theta^*}(x|Z = Z^{(i)})$, $\theta^* = \underset{\theta}{argmax} \prod_{i=1}^n P_\theta(x^{(i)})$

For computation, we approximate P_θ by $q_\phi(Z|x)$. Conditional probability $P_\theta(x|Z)$ is generative model. Approximate function $q_\phi(Z|x)$ is probabilistic encoder. $q_\phi(Z|x)$ should be close to $P_\theta(Z|x)$, so minimize $D_{KL}(q_\phi(Z|x)||P_\theta(Z|x))$

To encourage less blurring (Posterior collapse)
Use more of latent code ⇔ Disentangle latent representation:
①. Allow some Lagrangian weight between 2 ELBO terms (B-VAE).
②. introduce a mutual information term (Info-VAE).

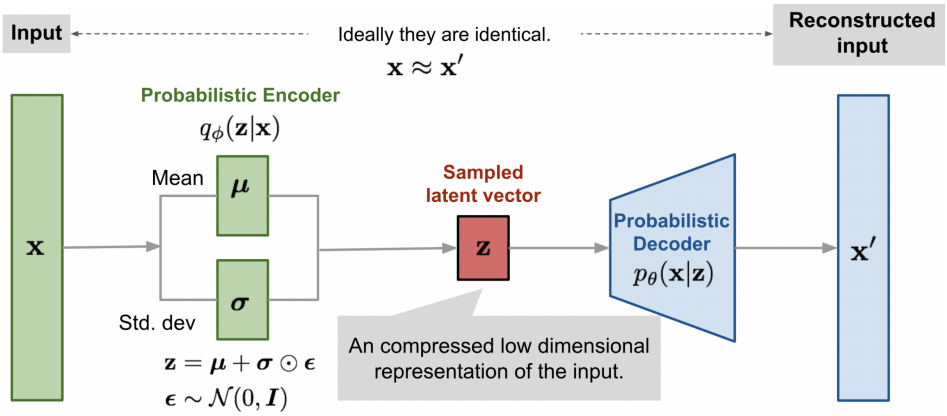
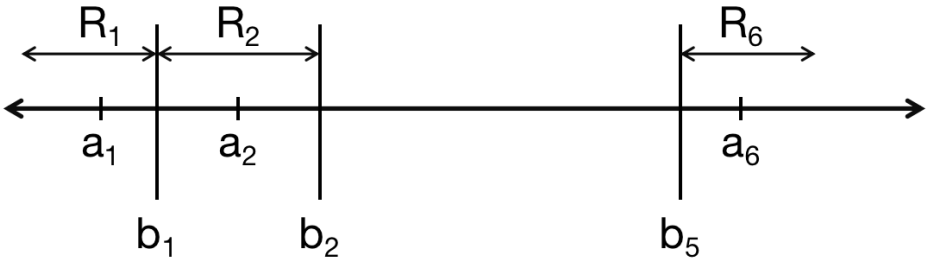


Illustration of variational autoencoder model with the multivariate Gaussian assumption.

2.11 Basics of quantization

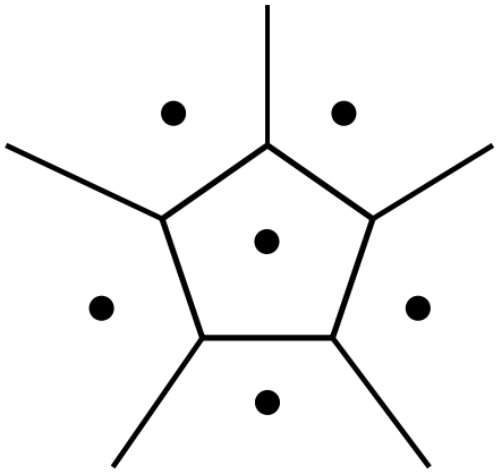
Constrain latent space to discrete.
Introduction to quantization: Consider iid sequence of analog rv $u_1, u_2 \sim P_U(u)$ quantizer maps this sequence into a sequence of discrete rv $v_1, v_2, ...$, where V_m should represent U_m for each m, with little distortion. If restrict to alphabet of size M, then V_M can't represent U_M perfectly on general, larger M implies less distortion.
$U_m \longrightarrow \boxed{\text{encoder}} \overset{I \leftarrow \{1,...,M\}}{\longrightarrow} \boxed{\text{decoder}} \longrightarrow V_m$
Mean-squared distortion is $\mathbb{E}[(u-v)^2]$



- Given representation points $\{a_j\}$, how should intervals $\{R_j\}$ chosen?
  Nearest neighbor condition, $b_j = \dfrac{a_j + a_{j+1}}{2}$
- Given intervals $\{R_j\}$, how should representation points $\{a_j\}$ be chosen?
  Centroid condition
  $a_j = \mathbb{E}[U|U \in R_j]$

Lloyd-Max algorithm:
Given M, $f_U(u)$;
①. Choose arbitrary initial set of M representation points, $a_1 > a_2 < ... < a_M$
②. for each j, $b_j = \dfrac{a_j + a_{j+1}}{2}$
③. for each j, set $a_j = \mathbb{E}[U|U \in [b_{j-1}, b_j]]$
④. Repeat ①,② until MSE doesn't change.

Vector quantization
Quantize n rv together. Region R_j must be set of points (u, u') that are closest to (a_j, a'_j) than to any other representation points.
Voronoi regions



Convex polygonal regions: boundaries are perpendicular bisectors between neighboring parts.

2.12 VQ-VAE approach

VQ advantages:
1. space filling; 2. shape; 3. memory
VAE

- encoder which parameterize approximate posterior $q(Z|x)$

- decoder $p(x|Z)$
- prior $p(Z)$

Define latent embedding space as subset of R^D with k representation points, $e_i \in \mathbb{R}^D, i = 1, ..., k$.
The encoder output $E(x) = Z_e$ goes through nearest neighbor lookup to match one of the k embedding vectors and then this input to decoder $D(.)$.
$Z_q(x) = quantize(E(x)) = e_k$, where $k = argmin\|E(x) - e_i\|_2$. For training, since argmin not differentiable in discrete case, gradient of loss function $\nabla_Z L$ from discrete input Z_q copy to encoder output.
Consider 3 terms in loss:
1. Reconstruction loss, which optimizes encoder/decoder
2. Due to copying from Z_e to Z_q, need something to train VQ
3. Ensure encoder commits to a representation

Let $sg(.)$ be stop gradient, defined as identity of forward computation and has zero partial derivatives:
$L = \underset{reconstruction\ loss}{\|x - D(e_k)\|_2^2} + \underset{VQ\ loss}{\|sg[E(x)] - e_k\|_2^2} + \beta \underset{constraint\ loss}{\|E(x) - sg(e_k)\|_2^2}$
Last piece is learn a prior in latent space $P(Z)$, so we can sample from it to do generation via decoders.

3 Lecture 7-9: generative adversarial networks (GANs)

Learning objectives:

- Explain the basic game-theoretic formulation of GANs
- Define a normal form game
- Describe the standard cost functions for generators and discriminators in GANs
- Describe the cost functions for generators and discriminators in GANs that lead to MLE
- Derive cost functions for generators and discriminators in GANs that lead to MLE
- Discuss stochastic gradient descent algorithms for GANs
- Describe the mode collapse and partial mode collapse phenomena and several ways to mitigate it
- Describe the basic formulation of Wasserstein GANs
- Derive the Kantorovich-Rubinstein duality in optimal transport theory
- Discuss f-GANs
- Describe GANs in the robust statistics framework
- Discuss novelty-quality tradeoffs in creativity
- Discuss ways that have been proposed to assess the performance of generative algorithms, especially GANs which only have implicit density estimation, and how this is related to interpolative and extrapolative generation

3.1 Basic game-theoretic formulation of GANs

One weakness of VAE is the variational bounds has a gap such that the model will not be consistent. GAN approach differently circumvents needs for variational bound $\Rightarrow$ NN models within GAN are universal approximators, and this enables a proof of asymptotic consistency in getting true objective distributions.
Training requires finding Nash equilibrium of a game, which is in general more difficult than optimizing an objective function.
Generator creates sampler that are intended to come from same distribution as training data
Discriminator examines samples to determine whether real or fake (supervised learning for binary classification).
In graphical model form of GAN, there will be observed variable X and latent variable Z. The players in game are represented by 2 functions, that one differentiable w.r.t their inputs and their parameters.
Discriminator is function D that takes x as input has parameters $\Theta^{(D)}$; generator is function that takes Z as input with parameters $\Theta^{(G)}$. Both players have cost functions that are defined in terms of both player's parameters.
Discriminator wants to minimize $J^{(D)}(\Theta^{(D)}, \Theta^{(G)})$, only controls $\Theta^{(D)}$. Generator wants to minimize $J^{(G)}(\Theta^{(D)}, \Theta^{(G)})$, only controls $\Theta^{(G)}$. Each player's cost depends on other player parameters $\Rightarrow$ Game, rather than optimization.

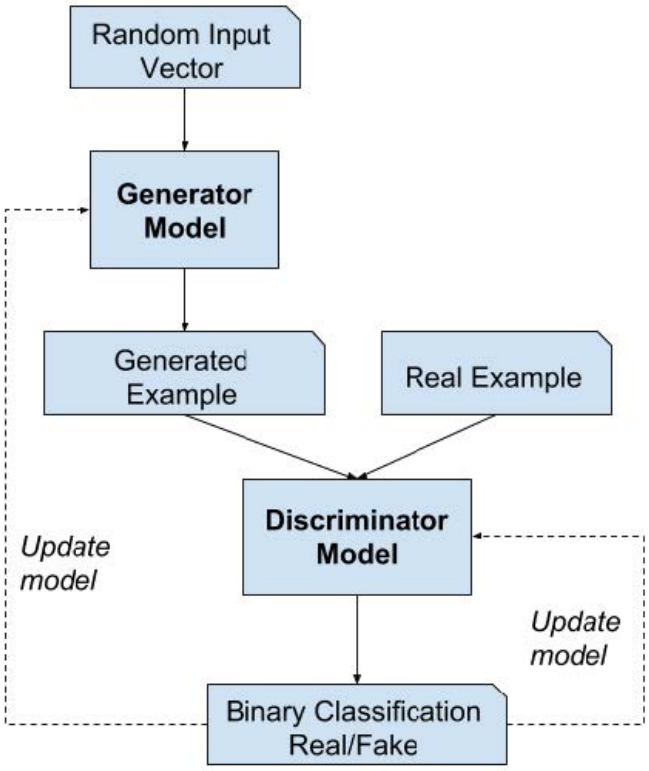


Figure 3: GAN architecture

Here a Nash Equilibrium is a tuple $(\Theta^{(D)}, \Theta^{(G)})$ is local minimum of $J^{(D)}$ w.r.t. $\Theta^{(D)}$ and is local minimum $J^{(G)}$ w.r.t $\Theta^{(G)}$.

3.2 Normal form game

An N-player normal form game consists of:
1. finite set of N players
2. strategy spaces for players, $S_1, S_2, ..., S_N$
3. payoff function for players, $U_i : S_1 \times S_2 \times ... \times S_N \rightarrow \mathbb{R}$

A natural representation of a two-player game is using a bi-matrix colum strategy.

	column strategy		
row strategy	$\begin{bmatrix}(u_1(r_1,c_1),u_2(r_1,c_1)) & (u_1(r_1,c_2),u_2(r_1,c_2)) & ... \\ (u_1(r_2,c_1),u_2(r_2,c_1)) & ... & ... \\ ... & ... & (u_1(r_m,c_n),u_2(r_m,c_n))\end{bmatrix}$		

where $s_1 = \{r_i, i = 1, 2, ..m\}$, $s_2 = \{c_j, j = 1, ..., n\}$
E.g.

(4,2)	(2,3)
(6,-1)	(0,0)

Find Nash equilibrium: run best response dynamics and see what happens.
Proofs of existence of Nash equilibrium come from fixed point theorems like Bronwer, Kakutari, etc.

3.3 Standard cost functions in GANs

Training process of GANs consists of simultaneous SGD ⇒ in each time step, two datasets sampled:
①. subset of x, training data
②. set of z values drawn from current models prior over latent variables.
Two simultaneous gradient steps:
One updates $\Theta^{(D)}$ to reduce $J^{(D)}$
One updates $\Theta^{(G)}$ to reduce $J^{(G)}$

Cost function for discriminator for binary classification.
$J^{(D)}(\Theta^{(D)}, \Theta^{(G)}) = -\frac{1}{2}\mathbb{E}_{x\sim P_{data}(x)} logD(x) - \frac{1}{2}\mathbb{E}_z log(1 - D(G(z)))$
Cross entropy cost to minimize to train standard binary classification with a sigmoid output.
By training discriminator, we can also estimate $\frac{P_{data}(x)}{P_{model}(x)}$. Estimating this ratio allows computing various divergences. Rather than lower bounds like ELBO, GAN approximation based on using supervised learning to estimate ratio of two densities.

What's cost function for generator?
Simplest approach is to require game to be zero sum, so sum of costs of all players is 0. $J^{(G)} = -J^{(D)}$.
So entire game can be summarized by value function specifying the discriminator's payoff:
$V(\Theta^{(D)}, \Theta^{(G)}) = -J^{(D)}(\Theta^{(D)}, \Theta^{(G)})$
Since zero sum games yield minmax characterization:
$\Theta^{(G)^*} = \underset{\Theta^{(G)}}{argmin}\ \underset{\Theta^{(D)}}{max} V(\Theta^{(D)}, \Theta^{(G)})$, where $\Theta^{(G)^*}$ corresponding to minimizing Jenson-Shanon (JS) divergence between data and model distribution.
An alternative heuristic that seems to work better in practice ⇒ still use cross entropy minimization for generator, but instead of flipping sign on discriminator, we flip the target used to construct cross entropy.
$J^{(G)} = -\frac{1}{2}\mathbb{E}_z logD(G(z))$.
So generator maximizing the log-probability of discriminator being mistaken rather than minimizing log probability of discriminator being correct.

3.4 Cost functions lead to MLE

MLE is minimizing KL divergence.
$\Theta^* = \underset{\theta}{argmin} D_{KL}(P_{data}(x)||P_{model}(x, \theta))$

If discriminator optimal, then we can minimize to obtain MLE when $J^{(G)} = -\frac{1}{2}\mathbb{E}_z \, exp(\sigma^{-1}(D(G(z))))$, where σ is logistic sigmoid.

minmax $\Leftrightarrow$ JS & MLE $\Leftrightarrow$ KL

Goals of discriminator is to minimize:

$J^{(D)}(\Theta^{(D)}, \Theta^{(G)}) = -\frac{1}{2}\mathbb{E}_{x\sim P_{data}(x)} log D(x) - -\frac{1}{2}\mathbb{E}_z log(1 - D(x))$ w.r.t $\Theta^{(D)}$

Value for $D(x)$ is specified for each value x to minimize $J^{(D)}$ w.r.t D. Write functional derivatives for a given x, set it zero: $\frac{\partial \, J^{(D)}}{\partial \, D(x)} = 0$.

Solving will yield $D^*(x) = \frac{P_{data}(x)}{P_{data}(x) + P_{model}(x)}$. Estimating it is the key to GANs.

3.5 SGD algorithm for GAN

MLE (KL divergence) in GAN (original GAN in minmax $\Leftrightarrow$ JS divergence). Derive a cost function that yields an approximate MLE. Want expected gradient of $J^{(D)}$ to match expected gradient of $D_{KL}(P_{data}||P_g)$. A gradient of $J^{(D)} = \mathbb{E}_{x\sim P_g} f(x)$ where we will choose $f(.)$ to match $\frac{\partial \, D_{KL}(P_{data}||P_g)}{\partial \, \theta}$

$\frac{\partial \, J^{(G)}}{\partial \, G} = \mathbb{E}_{x\sim p_g} f(x) \frac{\partial \, log(P_g(x))}{\partial \, G}$, where we assumes:

1. $P_g(x) \geq 0$ everywhere, so $P_{g(x)} = exp(log(P_g(x)))$
2. function derivative is continuous, so we can interchange derivative integral (Leibniz rule)

So we see derivatives of $J^{(G)}$ are close to what we want, just expectation is w.r.t samples from P_g rather that P_{data}, so pick $f(x) = \frac{P_{data}(x)}{P_g(x)}$ by importance sampling trick.

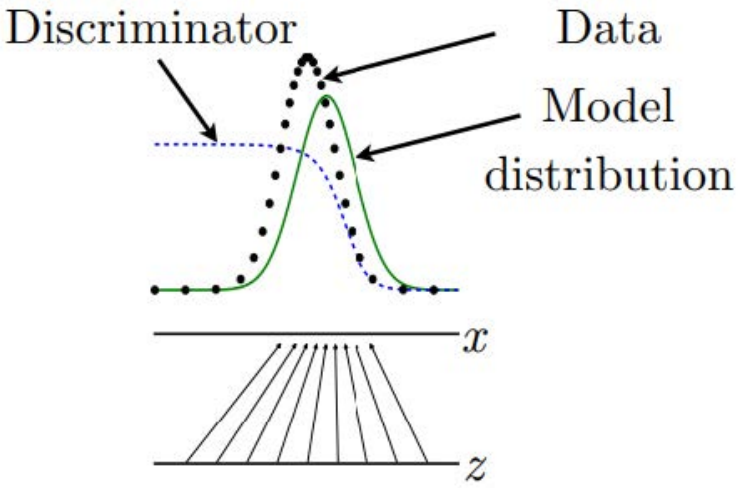


Figure 35: An illustration of how the discriminator estimates a ratio of densities. In this example, we assume that both z and x are one dimensional for simplicity. The mapping from z to x (shown by the black arrows) is non-uniform so that $p_{\text{model}}(x)$ (shown by the green curve) is greater in places where z values are brought together more densely. The discriminator (dashed blue line) estimates the ratio between the data density (black dots) and the sum of the data and model densities. Wherever the output of the discriminator is large, the model density is too low, and wherever the output of the discriminator is small, the model density is too high. The generator can learn to produce a better model density by following the discriminator uphill; each $G(z)$ value should move slightly in the direction that increases $D(G(z))$. Figure reproduced from Goodfellow *et al.* (2014b).

3.6 Model collapse

GAN is zero sum minmax game with ① discriminator ② generator. Most people think GAN successful if:
1. generator reliably generates data that fools discriminator
2. creates sample that are as diverse as distribution of real world

3.7 Model collapse

Fail to achieve ② by achieving ① through a concentrated distribution. Model collapse may arise because maxmin solution of GAN game is different from minmax solution.
When we find model $G^* = \min_G \max_D V(G, D)$, G^* will draw sample from the full distribution when we exchange order to have maxmin $G^* = \max_D \min_G V(G, D)$. Min of G is inner loop of optimization. The generator asked to map every Z value to the single output x that the discriminator believes is most to be real rather than fake.
Simultaneous gradient descent doesn't clearly privilege minmax over maxmin.
Solution methods:
Unrolling: updating generator's loss function to backpropagate through k steps of gradient updates for discriminator. Let generator see k steps into future to encourage more diverse samples.
Parking: modify discriminator to make decisions on several samples of same class, either real or fake. Seeing multiple identical cases is giveaway for fake.

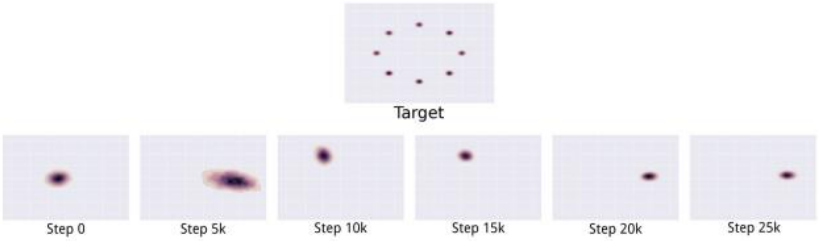


Figure 22: An illustration of the mode collapse problem on a two-dimensional toy dataset. In the top row, we see the target distribution p_{data} that the model should learn. It is a mixture of Gaussians in a two-dimensional space. In the lower row, we see a series of different distributions learned over time as the GAN is trained. Rather than converging to a distribution containing all of the modes in the training set, the generator only ever produces a single mode at a time, cycling between different modes as the discriminator learns to reject each one. Images from Metz *et al.* (2016).

3.8 Basic formulation WassersteinGAN

Wasserstein loss formulates loss functions to more directly represent minimizing distance between two probability distributions.
⇒ Make a winning turn in the game correlate with actually reducing distance between P_g and P_{data} rather than just fooling discriminator.
Recall JS divergence, $JS(P||Q) = \frac{1}{2}D_{KL}(P||M) + \frac{1}{2}D_{KL}(M||Q)$, where $M = (P+Q)/2$ was implicit objective in original GAN.
Now consider Wasserstein distance instead (optimal transportation theory):
$W(P, Q) = \inf_{\gamma \in \Pi(P,Q)} \mathbb{E}_{(x,y)\sim\gamma}[\|x - y\|]$, where $\Pi(P, Q)$ is a set of all joint distribution with marginal P, Q, $\gamma(x, y)$ is amount of mass that must be moved from x to y to convert P to Q.
Wasserstein distance is the cost of optimal transport map
⇒ continuous almost differentiable everywhere
⇒ JSD locally saturates as discriminator gets better, so gradients become zero and vanish.

3.9 Kantorovich-Rubinstein duality

Hard to handle inf, we can use Kantorovich-Rubirstein duality. $W(P,Q) = \sup_{\|f\|_L \leq 1} \mathbb{E}_{x\sim P}[f(x)] - \mathbb{E}_{y\sim Q}[f(y)]$, $W(P,Q)$ is supremum over all 1-Lipshitz functions.
K-Lipshitz continuity: given metric spaces $(x, dx), (y, dy)$, map $f : x \to y$ is k-Lipshitz if $dy(f(x_1), f(x_2)) \leq kdx(x_1, x_2)$ for all $(x_1, x_2) \in X$

Proof of K-R duality
Introduce Lagrangian multipliers $f, g : X \to R$ (bounded measurable):

$$\mathcal{L}(\gamma, f, g) = \int_{x\times x} \|x-y\|^2 \gamma(x,y)dydx - \int_x (P(x) - \int \gamma(x,y)dy)f(x)dx - \int_x (q(y) - \int (x,y)dx)g(y)dy$$
$$\mathcal{L}(\gamma, f, g) = \mathbb{E}_{x\sim P}[f(x)] + \mathbb{E}_{y\sim q}[g(y)] + \int_{x\times x} (\|x-y\|_2 - f(x) - g(y))\delta(x,y)dydx$$
$$W(P, P_g) = \inf_{\gamma} \sup_{f,g} L(\gamma, f, g) = \sup_{f,g} \inf_{\gamma} L(\gamma, f, g) \tag{2}$$

Note if $\|x-y\|_2 < f(x) + g(y)$ for some $x, y \in X$, then we can concentrate mass of γ at (x, y) and sent $\mathcal{L}(\gamma, f, g) \to -\infty$, so for all x, y, we must have $f(x) + g(y) \leq \|x-y\|_2$
With this constraint on Lagrangian, best we can do is minimize γ over $\gamma = 0$
$\sup_{f,g} \inf_{\gamma} \mathcal{L}(\gamma, f, g) = \sup_{f,g, f(x)+g(y)\leq\|x-y\|_2} [\mathbb{E}_{x\sim p}[f(x)] + \mathbb{E}_{y\sim q}[g(y)]] = W(p, q)$

Observe that optimizing over class of 1-Lipshitz function is lower bound on Wasserstein.
If h is 1-Lipshitz, $\mathbb{E}_{x\sim p}[h(x)] - \mathbb{E}_{y\sim q}[h(y)] = \int_{x\times x}(h(x) - h(y))\gamma(x,y)dxdy$
$\leq \int_{x\times x} \|x-y\|_2 \gamma(x,y)dxdy \leq W(p,q)$
$\sup_{\|h\|_L\leq 1} [\mathbb{E}_{x\sim p}[h(x) - \mathbb{E}_{y\sim q}[h(y)]]] \leq W(p,q)$
Wants to show this holds with equality. Consider a function $\mathcal{K}$ defined as:
$\mathcal{K} : x \to \inf_u [\|x-u\|_2 - g(u)]$, since g is bounded, inf is finite, and $\mathcal{K}$ is well-defined.
Claim: $\mathcal{K}$ is 1-Lipshitz
Proof: given $x, y \in X$. For any $u \in X$, by triangular inequality:
$\mathcal{K}(x) \leq \|x-u\|_2 - g(y) \leq \|x-y\|_2 + \|y-u\|_2 - g(u)$
This holds for any u.
$\mathcal{K}(x) \leq \|x-y\|_2 + \inf_u [\|x-u\|_2 - g(u)] = \|x-y\|_2 + \mathcal{K}(y)$
$\mathcal{K}(x) - \mathcal{K}(y) \leq \|x-y\|_2$, exchange x,y
$|\mathcal{K}(x) - \mathcal{K}(y)| \leq \|x-y\|_2$. It's 1-Lipshitz.
For any f, g that satisfy $f(x) + g(y) \leq \|x-y\|_2$,
$f(x) \leq \mathcal{K}(x) \leq \|x-x\|_2 - g(x) = -g(x)$
Plugin in: $\mathbb{E}_{x\sim p}[f(x)] + \mathbb{E}_{y\sim q}[g(y)] \leq \mathbb{E}_{x\sim p}[\mathcal{K}(x)] - \mathbb{E}_{y\sim q}[\mathcal{K}(y)]$
We conclude
$W(p,q) = \sup_{f,g,f(x)+g(y)\leq\|x-y\|_2} [\mathbb{E}_{x\sim p}[f(x)] + \mathbb{E}_{f\sim q}[g(y)]] \leq \sup_{\|h\|_2\leq 1} [\mathbb{E}_{x\sim p}[h(x)] - \mathbb{E}_{y\sim q}[h(y)]] \leq W(p,q)$
End of proof.
Objective of generator $\Rightarrow$ minimize Wasserstein distance rather than fooling discriminator.

3.10 f-GANs

Have different f-divergences as their objective.
Let P, Q be defined over the same space $P << Q$ (P is absolutely continuous w.r.t Q), then for a convex function $f : f(1) = 0$, f-divergence of P from Q:

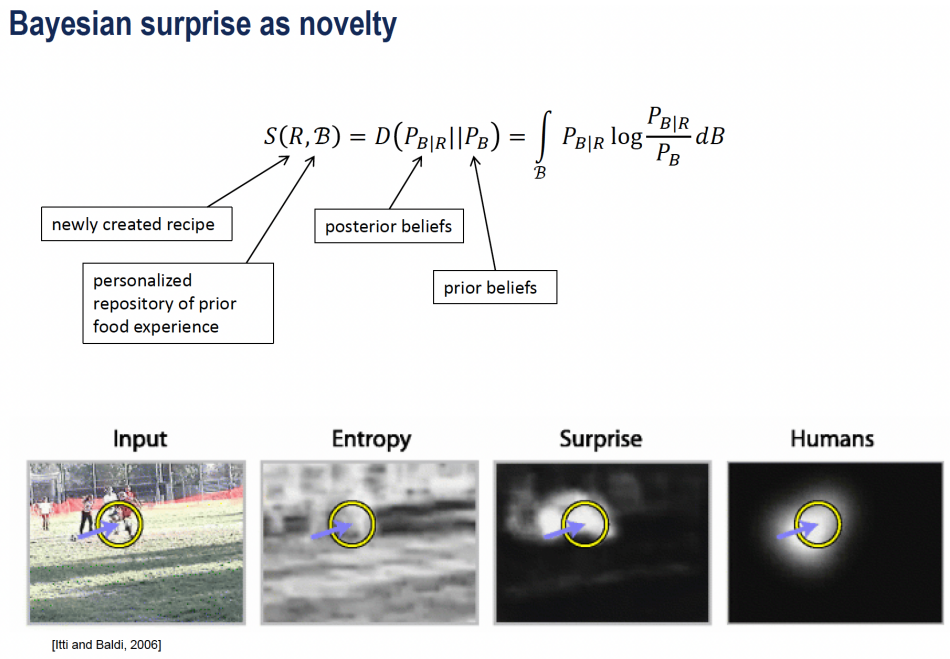
$D_f(P||Q) = \int f(\frac{dP}{dQ})dQ.$
When both absolutely continuous w.r.t a common reference measure (Lebesgue measure).
$D_f(P||Q) = \int f(\frac{p(x)}{q(x)})q(x)dx$
1. if $f(t) = tlog\ t \Rightarrow$ KL divergence.
2. if $f(t) = \frac{1}{2}[(t+1)log\ \frac{2}{t+1} + tlog\ t] \Rightarrow$ JS.
We can put in f-divergence as objective to define f-GAN. Match distribution as design idea, "density approximation".

3.11 Reformulation of GANs in robust statistics framework

Picking a generator $g(\cdot)$ from $\mathcal{G}$, minimize loss $\mathcal{L}(P_{g(z)}, P_x)$ over all $g \in \mathcal{G}$
$\mathcal{L}(P_{g(z)}, P_x)$ to be small. If $\mathcal{L}$ and $\mathcal{G}$ don't get right kind of small, try something else.
How to have P_x to be a slight perturbation of $g \in \mathcal{G}$ under $\mathcal{L}$ that make sense?

3.12 Novelty-quality tradeoffs in creativity

Creativity is the generation of an artifact that is judged to be novel and also to be appropriate, useful , or valuable by a suitably knowledge able social group.



3.13 Metrics to assess the performance of generative algorithms

Basic Tradeoff in Creativity: Average Case

Novelty-Quality tradeoff in Creativity

$$S(Q) = \max_{P_A(\alpha):E[q(A)]\geq Q} E[s(A,\Theta)]$$

Lemma [Varshney, 2013]

$$E[s(A,\Theta)] = I(A,\Theta).$$

Corollary

$$S(Q) = \max_{P_A(\alpha):E[q(A)]\geq Q} I(A,\Theta)$$

(Shannon's capacity-cost function)

A Note on the Inception Score

Shane Barratt [*1] Rishi Sharma [*1]

2.1. Desiderata

Before delving into the explanation of evaluation measures, first I list a number of desired properties that an efficient GAN evaluation measure should fulfill. These properties can serve as meta measures to evaluate and compare the GAN evaluation measures. Here, I emphasize on the qualitative aspects of these measures. As will be discussed in Section 3, some recent works have attempted to compare the meta measures quantitatively (*e.g.* computational complexity of a measure). An efficient GAN evaluation measure should:

1. favor models that generate high fidelity samples (*i.e.* ability to distinguish generated samples from real ones; discriminability),
2. favor models that generate diverse samples (and thus is sensitive to over-fitting, mode collapse and mode drop, and can undermine trivial models such as the memory GAN),
3. favor models with disentangled latent spaces as well as space continuity (*a.k.a* controllable sampling),
4. have well-defined bounds (lower, upper, and chance),
5. be sensitive to image distortions and transformations. GANs are often applied to image datasets where certain transformations to the input do not change semantic meanings. Thus, an ideal measure should be invariant to such transformations. For instance, score of a generator trained on CelebA face dataset should not change much if its generated faces are shifted by a few pixels or rotated by a small angle.
6. agree with human perceptual judgments and human rankings of models, and
7. have low sample and computational complexity.

In what follows, GAN measures will be discussed and assessed with respect to the above desiderata, and a summary will be presented eventually in Section 3. See Table 2.

4 Lecture 10-11: Auto regressive model

Learning objectives:

- Walk through Markov approximations of language
- Discuss generative grammar formulations of language and their contrast with Markov models
- Draw out autoregressive models parameterized by neural networks, including those with causal convolution layers
- Discuss MLE for autoregressive models

	Measure	Description
Quantitative	1. Average Log-likelihood [18, 22]	• Log likelihood of explaining realworld held out/test data using a density estimated from the generated data (*e.g.* using KDE or Parzen window estimation). $L = \frac{1}{N}\sum_i \log P_{model}(\mathbf{x}_i)$
	2. Coverage Metric [33]	• The probability mass of the true data "covered" by the model distribution $C := P_{data}(dP_{model} > t)$ with t such that $P_{model}(dP_{model} > t) = 0.95$
	3. Inception Score (IS) [3]	• KLD between conditional and marginal label distributions over generated data. $\exp\left(\mathbb{E}_{\mathbf{x}}\left[\mathbb{KL}\left(p\left(\mathrm{y} \mid \mathbf{x}\right) \parallel p\left(\mathrm{y}\right)\right)\right]\right)$
	4. Modified Inception Score (m-IS) [34]	• Encourages diversity within images sampled from a particular category. $\exp(\mathbb{E}_{\mathbf{x}_i}[\mathbb{E}_{\mathbf{x}_j}[(\mathbb{KL}(P(y\|\mathbf{x}_i)\|P(y\|\mathbf{x}_j))]])$
	5. Mode Score (MS) [35]	• Similar to IS but also takes into account the prior distribution of the labels over real data. $\exp\left(\mathbb{E}_{\mathbf{x}}\left[\mathbb{KL}\left(p\left(y \mid \mathbf{x}\right) \parallel p\left(y^{train}\right)\right)\right] - \mathbb{KL}\left(p\left(y\right) \parallel p\left(y^{train}\right)\right)\right)$
	6. AM Score [36]	• Takes into account the KLD between distributions of training labels vs. predicted labels, as well as the entropy of predictions. $\mathbb{KL}(p(y^{\text{train}}) \parallel p(y)) + \mathbb{E}_{\mathbf{x}}\left[H(y\|\mathbf{x})\right]$
	7. Fréchet Inception Distance (FID) [37]	• Wasserstein-2 distance between multi-variate Gaussians fitted to data embedded into a feature space $FID(r, g) = \|\mu_r - \mu_g\|_2^2 + Tr(\Sigma_r + \Sigma_g - 2(\Sigma_r\Sigma_g)^{\frac{1}{2}})$
	8. Maximum Mean Discrepancy (MMD) [38]	• Measures the dissimilarity between two probability distributions P_r and P_g using samples drawn independently from each distribution. $M_k(P_r, P_g) = \mathbb{E}_{\mathbf{x},\mathbf{x}'\sim P_r}[k(\mathbf{x},\mathbf{x}')] - 2\mathbb{E}_{\mathbf{x}\sim P_r,\mathbf{y}\sim P_g}[k(\mathbf{x},\mathbf{y})] + \mathbb{E}_{\mathbf{y},\mathbf{y}'\sim P_g}[k(\mathbf{y},\mathbf{y}')]$
	9. The Wasserstein Critic [39]	• The critic (*e.g.* an NN) is trained to produce high values at real samples and low values at generated samples $\hat{W}(\mathbf{x}_{test}, \mathbf{x}_g) = \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_{test}[i]) - \frac{1}{N}\sum_{i=1}^{N} f(\mathbf{x}_g[i])$
	10. Birthday Paradox Test [27]	• Measures the support size of a discrete (continuous) distribution by counting the duplicates (near duplicates)
	11. Classifier Two Sample Test (C2ST) [40]	• Answers whether two samples are drawn from the same distribution (*e.g.* by training a binary classifier)
	12. Classification Performance [1, 15]	• An indirect technique for evaluating the quality of unsupervised representations (*e.g.* feature extraction; FCN score). See also the GAN Quality Index (GQI) [41].
	13. Boundary Distortion [42]	• Measures diversity of generated samples and covariate shift using classification methods.
	14. Number of Statistically-Different Bins (NDB) [43]	• Given two sets of samples from the same distribution, the number of samples that fall into a given bin should be the same up to sampling noise
	15. Image Retrieval Performance [44]	• Measures the distributions of distances to the nearest neighbors of some query images (*i.e.* diversity)
	16. Generative Adversarial Metric (GAM) [31]	• Compares two GANs by having them engaged in a battle against each other by swapping discriminators or generators. $p(\mathbf{x}\|y=1; M_1)/p(\mathbf{x}\|y=1; M_2) = (p(y=1\|\mathbf{x}; D_1)p(\mathbf{x}; G_2))/(p(y=1\|\mathbf{x}; D_2)p(\mathbf{x}; G_1))$
	17. Tournament Win Rate and Skill Rating [45]	• Implements a tournament in which a player is either a discriminator that attempts to distinguish between real and fake data or a generator that attempts to fool the discriminators into accepting fake data as real.
	18. Normalized Relative Discriminative Score (NRDS) [32]	• Compares n GANs based on the idea that if the generated samples are closer to real ones, more epochs would be needed to distinguish them from real samples.
	19. Adversarial Accuracy and Divergence [46]	• Adversarial Accuracy. Computes the classification accuracies achieved by the two classifiers, one trained on real data and another on generated data, on a labeled validation set to approximate $P_g(y\|\mathbf{x})$ and $P_r(y\|\mathbf{x})$. Adversarial Divergence: Computes $\mathbb{KL}(P_g(y\|\mathbf{x}), P_r(y\|\mathbf{x}))$
	20. Geometry Score [47]	• Compares geometrical properties of the underlying data manifold between real and generated data.
	21. Reconstruction Error [48]	• Measures the reconstruction error (*e.g.* L_2 norm) between a test image and its closest generated image by optimizing for z (*i.e.* $min_{\mathbf{z}}\|\|G(\mathbf{z}) - \mathbf{x}^{(test)}\|\|^2$)
	22. Image Quality Measures [49, 50, 51]	• Evaluates the quality of generated images using measures such as SSIM, PSNR, and sharpness difference
	23. Low-level Image Statistics [52, 53]	• Evaluates how similar low-level statistics of generated images are to those of natural scenes in terms of mean power spectrum, distribution of random filter responses, contrast distribution, etc.
	24. Precision, Recall and F_score [23]	• These measures are used to quantify the degree of overfitting in GANs, often over toy datasets

- Derive the innovations form of random processes that meet the Paley-Wiener conditions and interpret in terms of convolutions
- Show the relationship between minimum mean-squared estimation and maximum entropy extrapolation
- Further discuss neural network-based autoregressive models

4.1 Markov approximations of language

4. GRAPHICAL REPRESENTATION OF A MARKOFF PROCESS

Stochastic processes of the type described above are known mathematically as discrete Markoff processes and have been extensively studied in the literature.[6] The general case can be described as follows: There exist a finite number of possible "states" of a system; $S_1, S_2, \ldots, S_n$. In addition there is a set of transition probabilities; $p_i(j)$ the probability that if the system is in state S_i it will next go to state S_j. To make this Markoff process into an information source we need only assume that a letter is produced for each transition from one state to another. The states will correspond to the "residue of influence" from preceding letters.

The situation can be represented graphically as shown in Figs. 3, 4 and 5. The "states" are the junction

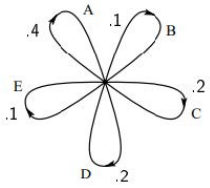


Fig. 3—A graph corresponding to the source in example B.

4.2 Generative grammar formulations of language

Context-free grammar

From Wikipedia, the free encyclopedia

In formal language theory, a **context-free grammar** (**CFG**) is a formal grammar whose production rules are of the form

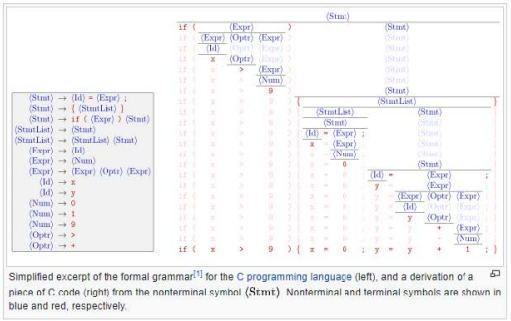
$$A \to \alpha$$

with A a *single* nonterminal symbol, and α a string of terminals and/or nonterminals (α can be empty). A formal grammar is "context free" if its production rules can be applied regardless of the context of a nonterminal. No matter which symbols surround it, the single nonterminal on the left hand side can always be replaced by the right hand side. This is what distinguishes it from a context-sensitive grammar.

A formal grammar is essentially a set of production rules that describe all possible strings in a given formal language. Production rules are simple replacements. For example, the first rule in the picture,

$$\langle \text{Stmt} \rangle \to \langle \text{Id} \rangle = \langle \text{Expr} \rangle ;$$

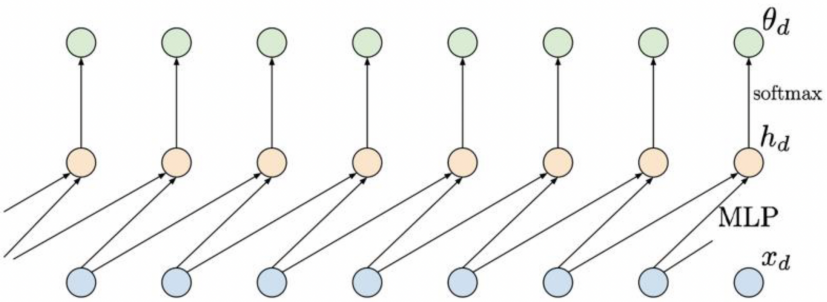
replaces $\langle \text{Stmt} \rangle$ with $\langle \text{Id} \rangle = \langle \text{Expr} \rangle ;$. There can be multiple replacement rules for a given nonterminal symbol. The language generated by a grammar is the set of all strings of terminal symbols that can be derived, by repeated rule applications, from some particular nonterminal symbol ("start symbol"). Nonterminal symbols are used during the derivation process, but do not appear in its final result string.



Simplified excerpt of the formal grammar[1] for the C programming language (left), and a derivation of a piece of C code (right) from the nonterminal symbol ⟨Stmt⟩. Nonterminal and terminal symbols are shown in blue and red, respectively.

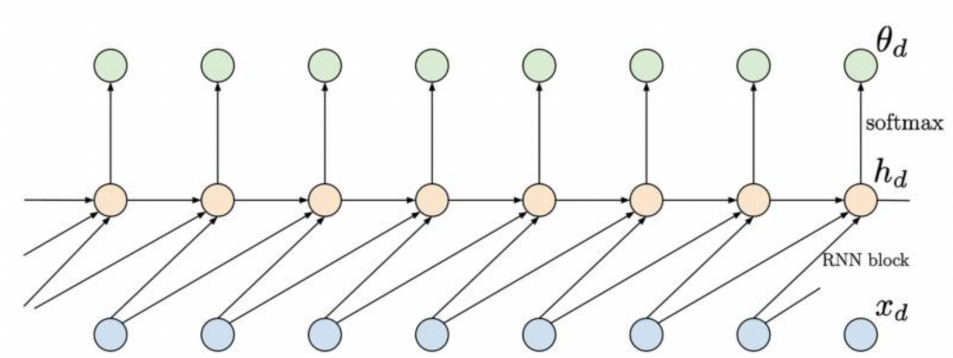
4.3 Autoregressive models parameterized by NN

$P(x, y) = P(x|y)P(y) = P(y|x)P(x)$, $P(x) = \sum_y P(x, y)$
Broadly, we want to model some $P(x)$. We hypothesize some factored form.
First, we generally write $P(x)$ for $x \in X^D$. e.g. $X = \{a, b, ..., z\}$ or $x = \{1, 2, ..., 256\}$.
$P(x) = P(x_1) \prod_{d=2}^{D} P(x_d|x_{<d})$, where $x_{<d} = \{x_1, ..., x_{d-1}\}$.
E.g. $P(x)$ for $D = 3$, $P(x_1)P(x_2|x_1)P(x_3|x_1, x_2)$. Modeling all the $P(x_d|x_{<d})$. Might be computationally burdensome.
We would have D separate models, of increasing complexity, as number of conditioning variables increases.
To help, we consider auto regressive models:
1. finite memory; 2. approximation; 3. shared parameters
$\Rightarrow$ NNs.
Finite memory:
E.g. D=3. $P(x_1)P(x_2|x_1) \prod_{d=3}^{D} P(x_d|x_{d-1}, x_{d-2})$, then also use a small multilayer perceptron to model the parameters. Single shared MLP to predict the probability for x_d.

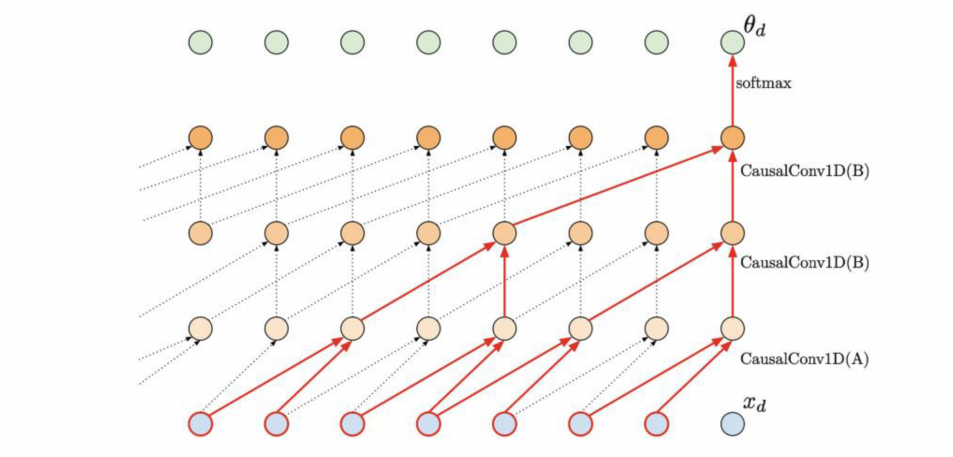


An example of applying a shared MLP depending on two last inputs. Inputs are denoted by blue nodes (bottom), intermediate representations are denoted by orange nodes (middle), and output probabilities are denoted by green nodes (top). Notice that a probability θ_d is not dependent on x_d

Long range memory:
Recurrent neural network. Make conditional distribution of form $P(x_d|x_{<d}) = P(x_d|RNN(x_{d-1}, h_{d-1}))$ where $h_d = RNN(x_{d-1}, h_{d-1})$ and h_d is hidden content that act as memory to capture long-range context.
They are sequential, slow in training/inference. Numerical issues in training them.
Convolutional neural network:
Can be used to model long-range dependencies rather than RNN.
Advantages:
1. kernels are shared (efficient parameterization)
2. processing in parallel (speedy computation)
3. stacking more layers allows effective kernel size to grow with network depth.



An example of applying an RNN depending on two last inputs. Inputs are denoted by blue nodes (bottom), intermediate representations are denoted by orange nodes (middle), and output probabilities are denoted by green nodes (top). Notice that compared to the approach with a shared MLP, there is an additional dependency between intermediate nodes h_d



An example of applying causal convolutions. The kernel size is 2, but by applying dilation in higher layers, a much larger input could be processed (red edges), thus, a larger memory is utilized. Notice that the first layers must be option A to ensure proper processing

4.4 Innovations form of random processes

One view of auto-regressive models is that they are predictors connect back to classic work in random process.

Discrete time random process: $x[n]$ is a sequence of random variables, defined for indices $n = -\infty, ..., 0, 1, 2, ...,$

$\Omega \to x[n]$, $\Gamma(z) = \dfrac{1}{L(z)}$

A discrete-time system is minimum phase if its system $L(z)$ and its inverse $\Gamma(z)$ are analytic in the exterior of unit disk $|z| > 1$.

A real wide-sense stationary digital process is regular if it's power spectrum:

$S(z) = \sum_{m=-\infty}^{\infty} R[m]z^{-m}$, where $R[m]$ is autocorrelation and this is Z transform. Can be written as a product: $S(z) = L(z)L(\frac{1}{z})$.

Denote by $l[n]$ and $\gamma[n]$, the delta response of $L(z)$ and $\Gamma(z)$. We can conclude that a regular process $x[n]$ is linearly equivalent with a whik-noise process $i[n]$

$i[n] = \sum_{k=0}^{\infty} \gamma[k]x[n-k]$, where $R_{ii} = \delta[m]$

$x[n] = \sum_{k=0}^{\infty} l[k]i[n-k]$, where $\mathbb{E}[x^2[n]] = \sum_{k=0}^{\infty} l^2[k] < \infty$

$x[n] \longrightarrow \boxed{\Gamma(z)} \xrightarrow{i[n]} \boxed{L(z)} \longrightarrow x[n]$

The process $i[n]$ is the innovation sequence of $x[n]$, and $\gamma(z)$ is its innovations filter. The whikening filter of $x[n]$ is $\Gamma(z) = \frac{1}{L(z)}$. H can be shown that the power spectrum $S(z)$ of a process $x[n]$. Can be factored as $S(z) = L(z)L(\frac{1}{z})$ if it satisfies Paley-Wipper condition (P-W condition).

P-W condition:

$\int_{-\pi}^{\pi} |logS(w)dw| < \infty$. If power spectrum $S(w)$ is integrable function, the P-W condition reduces to $\frac{1}{2\pi}\int_{-\pi}^{\pi} lnS(w)dw > -\infty$

The process $x[n]$ is called autoregressive (AR) if $\mathcal{L}(z) = \frac{b_0}{1 + a_1 z^{-1} + ... + a_N z^{-N}}$ and so process $x[n]$ satisfies recursion $x[n] + a_1 x[n-1] + ... + a_N x[n-N] = b_0 i[n]$ where $i[n]$ is whik-noise process.
The past $x[n-M]$ of $x[n]$ only depends on the past of $i[n]$ where $\mathbb{E}[i^2[n]] = 1$

4.5 Relationship between minimum mean-squared estimation and maximum entropy extrapolation

Estimation/prediction of AR processes:
Suppose $s[n]$ is an AR process of order M with autocorrelation $R[m]$. $\bar{S}[m]$ is some other general process with autocorrelation $\bar{R}[m]$ s.t. $\bar{R}[m] = R[m]$ for $|m| < M$.
The predictor (MSE optimizing) for these two process of order M will be identical, because they depend only on $R[m]$ for $|m| < M$.
Consider a class C_M of processes with identical autocorrelation for $|m| < M$, each $R[m]$ is an extrapolation of given data, and it can be shown the extrapolating sequence, obtained using maximum entropy method, is autocorrelation of an AR process of order M.
So maximum entropy extrapolation is autocorrelation of $S[n] \leftarrow C_m$ whose optimal predictor maximize the minimum MSE.
Max entropy $\Leftrightarrow$ MMSE prediction

4.6 Deep generative AR models

Think of x the thing we are modeling as categorical $x = \{1, ..., 256\}^{256 \times 256}$
We model $P(x)$ using the casual convolution-type architecture with many layers. Each conditional as follows:
$P(x_d|x_{<d}) = categorical(x_d|\theta_d(x_{<d})) = \prod_{l=1}^{L}(\theta_{d,l})^{[x_d=l]}$, where $[a=b]$ is Iverson bracket notation:

$$\begin{cases} [a=b] = 1 & if\ a = b \\ [a=b] = 0 & if\ a \neq b \end{cases}$$

and $\theta_d(x < d)$ is the output of the casual convolution layer with softmax in last layer, so $\sum_{l=1}^{L} \theta_{d,l} = 1$

What should be training objective?

we can maximize log-likelihood $ln\ P(D)$ where D is iid dataset of $\{x_1, ..., x_N\}$. So max $ln\ P(D)$:

$$\begin{aligned} ln\ P(D) &= ln \prod_n P(x_n) \\ &= \sum_n ln\ P(x_n) \\ &= \sum_n ln \prod_d P(x_{n,d}|x_{n,<d}) \\ &= \sum_n \sum_d^l nP(x_{n,d}|x_{n,<d}) \\ &= \sum_n (\sum_d ln\ Categorical(x_d|\theta_d(x_{<d}))) \\ &= \sum_n \sum_d (\sum_{l=1}^{L} [x_d = l] ln\ \theta_d(x_{<d})) \end{aligned}$$

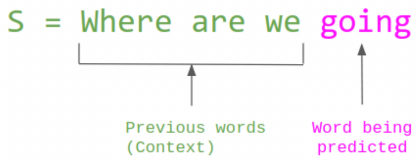
5 Lecture 12-13: Transformers

Learning objectives:

- Describe the basic approach of language modeling, the idea of self-supervision, and the cloze task
- Describe resource requirements and technologies behind large-scale language models
- Detail the attention mechanism in sequence-to-sequence models
- Detail the attention mechanism in language models
- Discuss the possibility of Transformers as universal computation engines
- Prove the universal approximation theorem for Transformers
- Use the allometric scaling framework
- Discuss scaling laws for Transformers

5.1 asic approach of language modeling

$$P_{(w_1,w_2,\ldots,w_n)} = p(w_1)p(w_2|w_1)p(w_3|w_1,w_2)...p(w_n|w_1,w_2,..,w_{n-1})$$
$$= \prod_{i=1}^{n} p(w_i|w_1,...,w_{i-1})$$



P(S) = P(Where) x P(are | Where) x P(we | Where are) x P(going | Where are we)

5.2 Resource requirements and technologies behind large-scale language models

We train on 140 GB of text drawing from a wide variety of domains: Wikipedia (En, De, Es, Fr), Project Gutenberg[1], submissions from 45 subreddits, OpenWebText[2], a large collection of news data (Hermann et al., 2015; Barrault et al., 2019; Sandhaus, 2008; Grusky et al., 2018), Amazon Reviews (McAuley et al., 2015), Europarl and UN data from WMT (En-De, En-Es, En-Fr) (Barrault et al., 2019), question-answer pairs (no context documents) from ELI5 (Fan et al., 2019) and the MRQA shared task[3], which includes the Stanford Question Answering Dataset (Rajpurkar et al., 2016), NewsQA (Trischler et al., 2016), TriviaQA (Joshi et al., 2017), SearchQA (Dunn et al., 2017), HotpotQA (Yang et al., 2018), and Natural Questions (Kwiatkowski et al., 2019). A full account of training data and associated control codes can be found in Table 7 in the Appendix.

We learn BPE (Sennrich et al., 2015) codes and tokenize the data using fastBPE[4], but we use a large vocabulary of roughly 250K tokens. This includes the sub-word tokens necessary to mitigate problems with rare words, but it also reduces the average number of tokens required to generate long text by including most common words. We use English Wikipedia and a 5% split of our collected OpenWebText data for learning BPE codes. We also introduce an `unknown` token so that during preprocessing we can filter out sequences that contain more than 2 unknown tokens. This, along with the compressed storage for efficient training (TFRecords) (Abadi et al., 2016), reduces our training data to 140 GB from the total 180 GB collected. Data was treated as a single stream of tokens with non-domain control codes inserted where appropriate (often at document boundaries).

5.3 Attention mechanism in sequence-to-sequence models

Deconstructing BERT: Distilling 6 Patterns from 100 Million Parameters
Attention in seq-to-seq models: Visualizing A Neural Machine Translation Model (Mechanics of Seq2seq Models With Attention)

5.4 Attention mechanism in transformers

: The Illustrated Transformer

5.5 Transformers as universal computation engines

Transformer as universal computation engine: Transformers as Universal over Domains

5.6 Universal approximation theorem for Transformers

Transformer blocks:

2 Transformer networks

A Transformer block is a sequence-to-sequence function mapping $\mathbb{R}^{d\times n}$ to $\mathbb{R}^{d\times n}$. It consists of two layers: a self-attention layer and a token-wise feed-forward layer, with both layers having a skip connection. More concretely, for an input $\boldsymbol{X} \in \mathbb{R}^{d\times n}$ consisting of d-dimensional embeddings of n tokens, a Transformer block with *multiplicative* or *dot-product* attention [Luong et al., 2015] consists of the following two layers[1]:

$$\mathrm{Attn}(\boldsymbol{X}) = \boldsymbol{X} + \sum_{i=1}^{h} \boldsymbol{W}_O^i \boldsymbol{W}_V^i \boldsymbol{X} \cdot \sigma[(\boldsymbol{W}_K^i \boldsymbol{X})^T \boldsymbol{W}_Q^i \boldsymbol{X}], \tag{1}$$

$$\mathrm{FF}(\boldsymbol{X}) = \mathrm{Attn}(\boldsymbol{X}) + \boldsymbol{W}_2 \cdot \mathrm{ReLU}(\boldsymbol{W}_1 \cdot \mathrm{Attn}(\boldsymbol{X}) + \boldsymbol{b}_1 \boldsymbol{1}_n^T) + \boldsymbol{b}_2 \boldsymbol{1}_n^T, \tag{2}$$

where $\boldsymbol{W}_O^i \in \mathbb{R}^{d\times m}$, $\boldsymbol{W}_V^i, \boldsymbol{W}_K^i, \boldsymbol{W}_Q^i \in \mathbb{R}^{m\times d}$, $\boldsymbol{W}_2 \in \mathbb{R}^{d\times r}$, $\boldsymbol{W}_1 \in \mathbb{R}^{r\times d}$, $\boldsymbol{b}_2 \in \mathbb{R}^d$, $\boldsymbol{b}_1 \in \mathbb{R}^r$, and $\mathrm{FF}(\boldsymbol{X})$ is the output of the Transformer block. The number of heads h and the head size m are two main parameters of the attention layer; and r denotes the hidden layer size of the feed-forward layer.

Transformer net:

We define the Transformer networks as the composition of Transformer blocks. The family of the sequence-to-sequence functions corresponding to the Transformers can be defined as:

$$\mathcal{T}^{h,m,r} := \{g : \mathbb{R}^{d\times n} \to \mathbb{R}^{d\times n} \mid g \text{ is a composition of Transformer blocks } t^{h,m,r}\text{'s}\}. \tag{3}$$

where $t^{h,m,r} : \mathbb{R}^{d\times n} \to \mathbb{R}^{d\times n}$ denotes a Transformer block defined by an attention layer with h heads of size m each, and a feed-forward layer with r hidden nodes.

We say that a function $f : \mathbb{R}^{d\times n} \to \mathbb{R}^{d\times n}$ is *permutation equivariant* if for any permutation matrix $\boldsymbol{P}$, we have $f(\boldsymbol{X}\boldsymbol{P}) = f(\boldsymbol{X})\boldsymbol{P}$; i.e., if we permute the columns of $\boldsymbol{X}$, then the columns of $f(\boldsymbol{X})$ are permuted in the same way. A Transformer block is permutation equivariant, which we formally prove in Section B. This consequently establishes the permutation equivariance of the class $\mathcal{T}^{h,m,r}$.

Claim 1. *A Transformer block $t^{h,m,r}$ defines a permutation equivariant map from $\mathbb{R}^{d\times n}$ to $\mathbb{R}^{d\times n}$.*

3 Transformers are universal approximators of seq-to-seq functions

In this section, we present our results showing that the Transformer networks are universal approximators of sequence-to-sequence functions. Let us start by defining the target function class $\mathcal{F}_{\mathrm{PE}}$, which consists of all continuous permutation equivariant functions with compact support that map $\mathbb{R}^{d\times n}$ to $\mathbb{R}^{d\times n}$. Here, continuity is defined with respect to any entry-wise ℓ^p norm, $1 \le p < \infty$. Given two functions $f_1, f_2 : \mathbb{R}^{d\times n} \to \mathbb{R}^{d\times n}$, for $1 \le p < \infty$, we define a distance between them as

$$\mathsf{d}_p(f_1, f_2) := \Big(\int \|f_1(\boldsymbol{X}) - f_2(\boldsymbol{X})\|_p^p \, d\boldsymbol{X} \Big)^{1/p}.$$

The following result shows that a Transformer network with a constant number of heads h, head size m, and hidden layer of size r can approximate any function in $\mathcal{F}_{\mathrm{PE}}$.

Theorem 2. *Let $1 \le p < \infty$ and $\epsilon > 0$, then for any given $f \in \mathcal{F}_{\mathrm{PE}}$, there exists a Transformer network $g \in \mathcal{T}^{2,1,4}$, such that $\mathsf{d}_p(f, g) \le \epsilon$.*

3.1 Transformers with trainable positional encodings

In order to endow the Transformer networks with the ability to capture the information about the position of tokens in the input sequence, it is a common practice to add positional encodings $\boldsymbol{E} \in \mathbb{R}^{d\times n}$ to the input sequence before feeding it to the Transformer network [Vaswani et al., 2017, Devlin et al., 2018]. Consider the functions represented by Transformers with positional encodings:

$$\mathcal{T}_{\mathrm{P}}^{h,m,r} := \{g_{\mathrm{P}}(\boldsymbol{X}) = g(\boldsymbol{X} + \boldsymbol{E}) \mid g \in \mathcal{T}^{h,m,r} \text{ and } \boldsymbol{E} \in \mathbb{R}^{d\times n}\}. \tag{4}$$

Here we show that if $\boldsymbol{E}$ is trainable, these positional encodings are sufficient to remove the permutation equivariance restriction of the Transformers. Towards this, we define $\mathcal{F}_{\mathrm{CD}}$ to be the set of all continuous functions that map a compact domain in $\mathbb{R}^{d\times n}$ to $\mathbb{R}^{d\times n}$. Note that $\mathcal{F}_{\mathrm{CD}}$ does not have the restriction of permutation equivariance as in $\mathcal{F}_{\mathrm{PE}}$, but any $f \in \mathcal{F}_{\mathrm{CD}}$ is defined on a compact domain instead of the whole $\mathbb{R}^{d\times n}$. The following result states that, equipped with the trainable positional encodings, Transformers can approximate any sequence-to-sequence function in $\mathcal{F}_{\mathrm{CD}}$.

Theorem 3. *Let $1 \le p < \infty$ and $\epsilon > 0$, then for any given $f \in \mathcal{F}_{\mathrm{CD}}$, there exists a Transformer network $g \in \mathcal{T}_{\mathrm{P}}^{2,1,4}$ such that we have $\mathsf{d}_p(f, g) \le \epsilon$.*

4 Conclusion

In this paper, we prove that Transformer networks are universal approximators of any continuous and permutation equivariant sequence-to-sequence functions, which shed light on the expressive power of Transformer networks. We also theoretically validate the use of additive positional encodings in Transformers, as they can remove the permutation equivaraince restriction and make Transformers universal approximators of arbitrary continuous sequence-to-sequence functions.

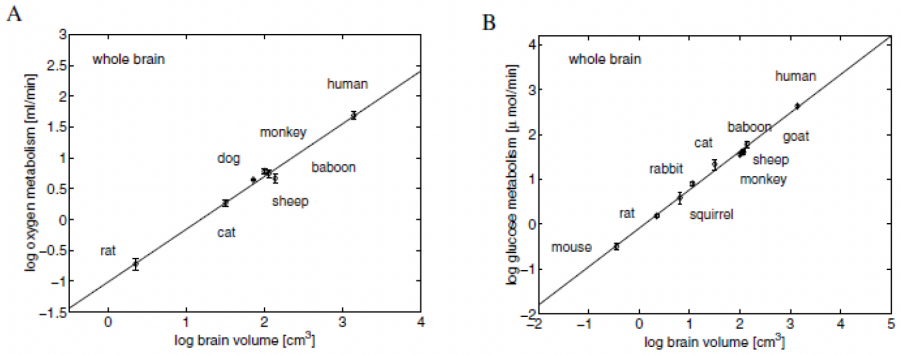
In the supplementary material, we present the proofs of our theorems, which reveal that self-attention layers in Transformer networks can compute *contextual mappings*; this is one of the crucial components that make Transformer networks universal. We also discuss and experiment with other simpler layers that can implement weaker forms of contextual mappings.

5.7 Allometric scaling

In neurobiology, one can look at allometric scaling relationships:

- across different species with similar brain architectures [evolution],
- scaling relationships for different individuals of same species [growth],
- properties of the brain within the same individual [structure]

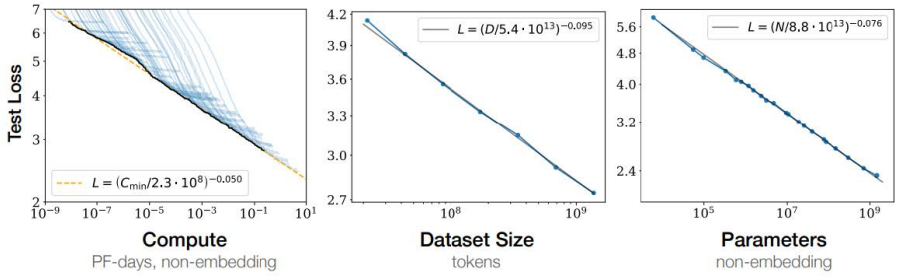
The relationship between the two measured quantities is usually expressed as a power law equation: $y = kx^{\alpha}$, where α is the scaling exponent of the law.
How should we interpret superlinear ($\alpha > 1$)) or sublinear ($\alpha < 1$) scaling?



Scaling of the total basal cerebral metabolism with brain volume. The least-square fit line for the log – log plot yields the following. (**A**) For the total oxygen consumption rate, the scaling exponent was 0.86 ± 0.04 (y = 0.86x - 1.02, R^2 = 0.989, p < 10^{-4}, *n* = 7), and its 95% confidence interval was 0.75 to 0.96. (**B**) For the total glucose utilization rate, an identical exponent 0.86 ± 0.03 was found (y = 0.86x - 0.09, R^2 = 0.994, p < 10^{-4}, *n* = 10) and its 95% confidence interval was 0.80 to 0.91.

5.8 Scaling law for transformers

Model performance depends most strongly on scale, which consists of three factors: the number of model parameters N (excluding embeddings), the size of the dataset D, and the amount of compute C used for training. Within reasonable limits, performance depends very weakly on other architectural hyperparameters such as depth vs. width.
Performance has a power law relationship with each of the three scale factors N, D, C when not bottlenecked by the other two, with trends spanning more than six orders of magnitude



Universality of overfitting : Performance improves predictably as long as we scale up N and D in tandem, but enters a regime of diminishing returns if either N or D is held fixed while the other increases. The performance penalty depends predictably on the ratio $N^0.74/D$, meaning that every time we increase the model size 8x, we only need to increase the data by roughly 5x to avoid a penalty.
Universality of training : Training curves follow predictable power laws whose parameters are roughly independent of the model size. By extrapolating the early part of a training curve, we can roughly predict the loss that would be achieved if we trained for much longer.
Transfer improves with test performance : When we evaluate models on text with a different distribution than they were trained on, the results are strongly correlated to those on the training validation set with a roughly constant offset in the loss in other words, transfer to a different distribution incurs a constant penalty but otherwise improves roughly in line with performance on the training set.
Sample efficiency : Large models are more sample efficient than small models, reaching the same level of performance with fewer optimization steps and using fewer data points.
Convergence is inefficient : When working within a fixed compute budget C but without any other restrictions on the model size N or available data D, we attain optimal performance by training very large models and stopping significantly short of convergence. Maximally compute efficient training would therefore be far more sample efficient than one might expect based on training small models to convergence, with data requirements growing very slowly as $D \sim C^{0.27}$ with training compute.

6 Lecture 14. Neural text decoding and prompt programming

Learning objectives
$\longrightarrow$ Prompt programming $\longrightarrow$ Generative AI models $\longrightarrow$ Neural decoding $\longrightarrow$

Theory of mind:
1. Understanding of your creative partners
2. recent experiments show "theory of mind" abilities improve human-AI creativity.

Formulism for prompt programming / prompt engineering.
Reynolds & McDonrel (ACM CHI, 2021)
Prompting using natural human language (principles):
1. direct specification

2. examples
3. memetic proxy
4. constraining behavior
5. closed-ended questions
6. metaprompt programming
Backpropagating through large language model to find the prompt that yields it.

Neural decoding:
Language model gives posterior probability for next token (trained on "predict next token" task).
e.g. "This sentence will end _?". How?
1. sample from posterior
2. forward-backward (beam search)
3. grammar-based principle
4. mode
5. moment of the induced next posterior (from top-k) [Basu et.al 2021], mirostat control perplexity, surprise
6. sample from top-k
7. (prompt programming)
8. top-p sampling (nucleus sampling) → other moment-based techniques