CS 182 Midterm 1 — Jeffry Shen

②③④ Basic Principles

Setup: x_i, y_i: input, labels — Problem: $\hat{y} = f_\theta(x)$

$f_\theta(\cdot)$ model

$l(y, \hat{y})$ loss function — optimal $\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} l(y_i, f_\theta(x_i))$

Neural Nets: differentiable operations allow nonlinearities

1) Expressivity: express patterns we want to learn $f_\theta(\cdot)$
2) Reliably learnable

Represent piecewise linear func that are differentiable using elbows

weight w: slope $= w$

located at bias b: $-\frac{b}{w}$

Initialization: He initialization: Gaussian $N(0, \frac{2}{d})$

Regularization: to prevent overfitting

1) Explicit Regularization: add regularizer term

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \left[\frac{1}{n} \sum_{i=1}^{n} l_{train}(y_i, f_\theta(x_i)) + R(\theta)\right]$$

ridge regularization ($R(\theta) = \lambda \|\theta\|^2$) prevents θ too large

OLS Ridge Regression: $\hat{\vec{w}} = (X^T X + \lambda I)^{-1} X^T \vec{y}$

GD Ridge Regression: gradient: $\eta(2X^T(\vec{y} - X\hat{\vec{w}}) - 2\lambda \hat{\vec{w}})$

update: $\vec{w}_{(t+1)} = (1 + 2\eta\lambda)\vec{w}_{(t)} + \eta 2X^T(\vec{y} - X\hat{\vec{w}}_{(t)})$

2) Data Augmentation: fake observations of features

3) Implicit Regularization: optimizer as implicit regularizer

Maximum A Posteriori (MAP): $R(\theta)$ responds to a prior

Gradient Descent (GD): iterative approach to find local optima

Errors:
1) Irreducible error: noise or randomness from $Y|X$
2) Approximation error: not flexible enough for true signal
3) Estimation error: bias: systematic error of learning; variance: randomness of training process

Features: representation of data-driven, allow generalized linear model to work well

⑤⑥ Survey of Architectures & Problems

Network Architectures:

1) Multi-layer Perceptron (MLP): fully connected
2) Convolution Neural Nets (CNNs): spatial regularity embedded, ↑ images
3) Recurrent Neural Nets (RNN): CNNs w internal state over time
4) Graph Neural Nets (GNN): nearby items more related
5) Transformers: access input data elsewhere and weight share

Types of Problems:

1) Regression 2) Classification 3) Generation 4) Recommendation

Optimization:

1) Gradient Descent: descend using learning rate. To ensure grad descent stable, $\eta < \frac{1}{\sigma^2}$
2) Momentum based methods: low pass filter to make learning rate bigger w/o trouble for large singular values. LPF the directions that oscillate, update as avg grad direction

"Vanilla" Momentum $\nabla L(w_t)$

"Nesterov" Momentum $\nabla L(w_t - \eta(1-\beta) a_t)$

3) Adaptive approaches: change learning rates according to singular val

$\hat{a}_{k+1} = \frac{\vec{a}_{k+1}}{1 - \beta^k}$ $\quad \hat{V}_{k+1} = \frac{\vec{V}_{k+1}}{1 - (\beta')^k}$ $\quad w_{k+1}[i] = w_k[i] - \eta \frac{\hat{a}_{k+1}[i]}{\sqrt{\hat{V}_{k+1}[i]} + \varepsilon}$

⑦⑧⑨ CNN

class of neural nets used to analyze images

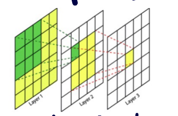
- respect "locality", "invariances",
- support hierarchical structure, multi-resolution understanding
- weight sharing: neighborhood of pixels is processed by kernel

Pooling 2x2 elements w stride to desired pixel

Output: $\frac{W - K + 2P}{S} + 1$

Receptive field: pixels in original image the output depends on

Receptive fields grow linearly but exponentially w stride / padding

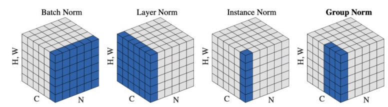


Data Augmentation: transfer domain knowledge into system

ex) autocontrast, rotation, translation

Standardization & Normalization: convert data to zero mean & unit variance, so raw magnitude doesn't have effect on gradient

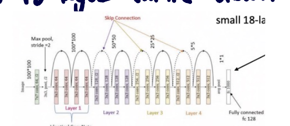
1) Batch Normalization: normalizes layer for mini batch
2) layer normalization: normalize across all channels
3) Instance normalization: each channel each training image
4) Group normalization: over group of channels



⑩ CNN Architectures, Dropout

Residual Network: skip connections, output to layer further down

ex) ResNet, allowed more expressive networks, learn salient features

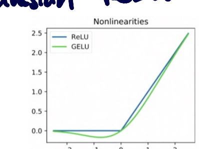


ConvNeXt: layer normalization + Gaussian ReLU

Gaussian ReLU (GeLU): $x \cdot \Phi(x)$

$\Phi(x)$: Gaussian CDF

- smoother, non-convex, non-monotonic



Depthwise Convolution: broken into channels, convolve, concat

Dropout: "ensemble-like" behavior to promote diversity and redundancy, learning rate can compensate for training slower

- kill certain units setting to 0

Stochastic Depth Regularization: drop entire residual blocks

Label Smoothing: probabilities in goal array

$$y = \left[\frac{1 - \alpha \cdot k - 1}{k}, \frac{\alpha}{k}, \frac{\alpha}{k}, \ldots\right]^T \quad k = \text{classes}$$

⑪ GNN

Graph w information in nodes, "generalized" CNN

GNN vs. CNN: diff number of neighbors in GNN, no ordering

Weight Sharing: consider itself and neighbor nodes, regardless of ordering and have learnable func $f_w[me, \sum_{neighbors} s_{w_2}(me, them) g_{w_3}(them)]$

Pooling: can cluster and pool or not as useful

Can run if we have lots of different graphs as data in training

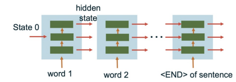
(12) RNN

Residual Connection blocks issue of vanishing gradients

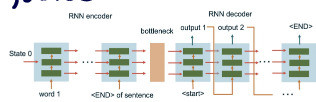
Speed Up Training by:

1) larger batch size, linearly scale learning rate
2) distributed data-parallel training
3) regularize by aggressive data augmentation

(14) Attention / Self-Supervision



RNN encoder: good for sequential words, shares same weight



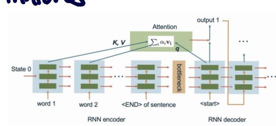
Bottleneck: include input statement

Attention: allow to look back at words originally embedded, allow info to flow along layers

use hash table to store key, values,

To get values for query, scan for closest matches of queries to keys

$e_i = \frac{q^T k_i}{\sqrt{d}}$

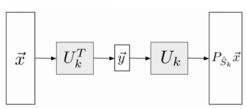


To track order information, we can use vector expression (complex expression)

(15) Self-Supervision and Autoencoders

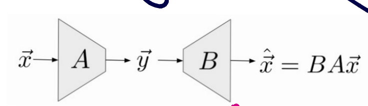
Self-supervision: for unsupervised learning (PCA, clustering), design to use loss func, grad to supervised

Easier to obtain unlabeled data, use dimensionality reduction clustering



PCA dimensionality reduction

→



learn $L(\vec{x}, \hat{\vec{x}}) = \frac{1}{n}\sum_{i=1}^{n} \|\vec{x}_i - BA\vec{x}_i\|^2$

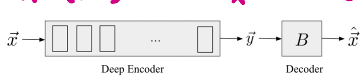
Autoencoders: want grad descent to learn $\vec{x}_i$ to $P_{S_k}\vec{x}_i$

doesn't necessarily need to be linear, k-dimensional structure

can replace A,B w nonlinear encoder/decoder

A Weight Sharing: $A = (B^TB)^{-1}B^T$ $\hat{\vec{x}} = BA\vec{x} = B(B^TB)^{-1}B^T\vec{x}$

Parametrization 4:



Data Augmentation can help prevent learning identity

Other

Beam Search Runtime: time $O(TkM\log M)$ & memory $O(T^2kM\log M)$

LSTM:

$C_t \to \oplus \to \oplus \to C_{t+1}$

f_t: sigmoid $(Wx_t + Wh_t + W_3(t + \text{bias}))$

input: tanh $(Wx_t + Wh_t + W_3(t + \text{bias}))$

Practice Exams

- Newton's Method can converge to global optimum when loss func optimized is convex
- SGD does not find same empirical gradient
- batch normalization introduces dependence between data pts in one mini batch
- Convolutional block reduces # of channels to speed up forward / backward

Misc

Linear Algebra

L2 Norm (Euclidean): $\|x\|_2 := \sqrt{\sum_{i=1}^{n} x_i^2} = \sqrt{x^Tx}$

L1 Norm: $\|x\|_1 = \sum_{i=1}^{n} |x_i|$

L∞ Norm: $\|x\|_\infty := \max_{1\le i\le n} |x_i|$

Frobenius Norm: $\|A\|_F := \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} A_{ij}^2} = \sqrt{Tr(A^TA)}$

Orthonormal: $u_i^Tu_j = 0$, $i \neq j$ and $\|u_i\|_2 = 1$

Trace: $Tr A = \sum_{i=1} A_{ii}$ sum of diagonal elements

Cauchy-Schwartz Inequality: $|z^Ty| \le \|y\|_2 \cdot \|z\|_2$

Fundamental Theorem of Linear Algebra

Range of a matrix is the orthogonal complement of the nullspace of its transpose

$R(A)^\perp = N(A^T)$

Spectral Theorem (Symmetric eigenvalue decomposition (SED))

$A = \sum_{i=1}^{n} \lambda_i u_i u_i^T = U\Lambda U^T$, $\Lambda = diag(\lambda_1, \dots, \lambda_n)$

$Tr\,\Sigma = \frac{1}{m}XX^T = \frac{1}{m}\|X\|_F^2$

Total Variance: $Tr\,\Sigma = Tr(U\Lambda U^T) = Tr(U^TU\Lambda) = Tr\,\Lambda = \lambda_1 + \dots + \lambda_n$

rank: linearly independent columns $X: \mathbb{R}^{m\times n}$

Rank-Nullity Theorem: $n - \dim(\text{nullspace}(X)) = \text{rank}(A)$

$\dim(\text{rowspace}(X)) = \dim(\text{columnspace}(X^T)) = \text{rank}(X^T) = \text{rank}(X)$

- symmetric matrix has real eigenvalues
- eigenvalues neg if concave
- Axis scaled by square roots of eigenvalues of Σ
- Covar $= \frac{XX^T}{n}$ $X \in \mathbb{R}^{n\times d}$

Symmetric matrix M

Positive Definite: if $w^TMw > 0$ all $w \neq 0$ ⇔ pos eigenvalues

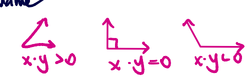
Positive Semidefinite: if $w^TMw \ge 0$ all w ⇔ nonnegative eigenvalues

Indefinite: if pos & neg eigenvalue

Invertible: no zero eigenvalue

convex: x^2 ∪

concave: $\sqrt{x}$



$\cos\theta = \frac{x\cdot y}{\|x\|\,\|y\|} = \underbrace{\frac{x}{\|x\|}}_{\text{length } 1} \cdot \underbrace{\frac{y}{\|y\|}}_{\text{length } 1}$

$E[\cos\theta] = 0$ $SD(\cos\theta) \propto \frac{1}{\sqrt{d}}$

$Tr(AB) = Tr(BA)$

$(A^T)^T = A$

$(A+B)^T = A^T + B^T$

$(AB)^T = B^TA^T$

PSD $Q = PDP^T$
$PD^{1/2}D^{1/2}P^T$
$= UU^T$

$A = A^{1/2}A^{1/2}$
$= (PD^{1/2}(P^T))(PDP^T) = PDP^T$

Eigenvalues 2x2

$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$ $m \pm \sqrt{m^2 - p}$

$m = \frac{a+d}{2}$ $p = \det(M)$ ad-bc

Probability

Bayes Rule: $P(Y=1|X) = \frac{P(X|Y=1)P(Y=1)}{P(X)}$ (posterior: $P(Y=1|X)$; prior: $P(Y=1)$)

$P(A,B) = \sum_C P(A,B,C)$

Chain Rule: $P(A,B,C) = P(A,B|C)P(C) = P(A|B,C)P(B|C)P(C)$

A conditionally independent given C: $P(A,B|C) = P(A|C)P(B|C)$

A independent of B given C: $P(A|B,C) = P(A|C)$

A,B independent: $P(A,B) = P(A)P(B)$

$P(A|B,C) = \frac{P(A,B,C)}{P(B,C)} = \frac{P(A,B|C)}{P(B|C)}$

$P(x) = P(x|Y=1)P(Y=1) + P(x|Y=-1)P(Y=-1)$

Matrix Derivatives

$$\frac{\partial \mathbf{x}^T\mathbf{a}}{\partial \mathbf{x}} = \frac{\partial \mathbf{a}^T\mathbf{x}}{\partial \mathbf{x}} = \mathbf{a}$$

$$\frac{\partial \mathbf{a}^T\mathbf{X}\mathbf{b}}{\partial \mathbf{X}} = \mathbf{a}\mathbf{b}^T$$

$$\frac{\partial \mathbf{a}^T\mathbf{X}^T\mathbf{b}}{\partial \mathbf{X}} = \mathbf{b}\mathbf{a}^T$$

$$\frac{\partial \mathbf{a}^T\mathbf{X}\mathbf{a}}{\partial \mathbf{X}} = \frac{\partial \mathbf{a}^T\mathbf{X}^T\mathbf{a}}{\partial \mathbf{X}} = \mathbf{a}\mathbf{a}^T$$

$$\frac{\partial \mathbf{X}}{\partial X_{ij}} = \mathbf{J}^{ij}$$

$$\frac{\partial (\mathbf{X}\mathbf{A})_{ij}}{\partial X_{mn}} = \delta_{im}(\mathbf{A})_{nj} = (\mathbf{J}^{mn}\mathbf{A})_{ij}$$

$$\frac{\partial (\mathbf{X}^T\mathbf{A})_{ij}}{\partial X_{mn}} = \delta_{in}(\mathbf{A})_{mj} = (\mathbf{J}^{nm}\mathbf{A})_{ij}$$

$$\frac{\partial}{\partial X_{ij}} \sum_{klmn} X_{kl}X_{mn} = 2\sum_{kl} X_{kl}$$

$$\frac{\partial \mathbf{b}^T\mathbf{X}^T\mathbf{X}\mathbf{c}}{\partial \mathbf{X}} = \mathbf{X}(\mathbf{b}\mathbf{c}^T + \mathbf{c}\mathbf{b}^T)$$

$$\frac{\partial (\mathbf{B}\mathbf{x}+\mathbf{b})^T\mathbf{C}(\mathbf{D}\mathbf{x}+\mathbf{d})}{\partial \mathbf{x}} = \mathbf{B}^T\mathbf{C}(\mathbf{D}\mathbf{x}+\mathbf{d}) + \mathbf{D}^T\mathbf{C}^T(\mathbf{B}\mathbf{x}+\mathbf{b})$$

$$\frac{\partial (\mathbf{X}^T\mathbf{B}\mathbf{X})_{kl}}{\partial X_{ij}} = \delta_{lj}(\mathbf{X}^T\mathbf{B})_{ki} + \delta_{kj}(\mathbf{B}\mathbf{X})_{il}$$

$$\frac{\partial (\mathbf{X}^T\mathbf{B}\mathbf{X})}{\partial X_{ij}} = \mathbf{X}^T\mathbf{B}\mathbf{J}^{ij} + \mathbf{J}^{ji}\mathbf{B}\mathbf{X} \qquad (\mathbf{J}^{ij})_{kl} = \delta_{ik}\delta_{jl}$$

$$\frac{\partial \mathbf{x}^T\mathbf{B}\mathbf{x}}{\partial \mathbf{x}} = (\mathbf{B}+\mathbf{B}^T)\mathbf{x}$$

$$\frac{\partial \mathbf{b}^T\mathbf{X}^T\mathbf{D}\mathbf{X}\mathbf{c}}{\partial \mathbf{X}} = \mathbf{D}^T\mathbf{X}\mathbf{b}\mathbf{c}^T + \mathbf{D}\mathbf{X}\mathbf{c}\mathbf{b}^T$$

$$\frac{\partial}{\partial \mathbf{X}}(\mathbf{X}\mathbf{b}+\mathbf{c})^T\mathbf{D}(\mathbf{X}\mathbf{b}+\mathbf{c}) = (\mathbf{D}+\mathbf{D}^T)(\mathbf{X}\mathbf{b}+\mathbf{c})\mathbf{b}^T$$

(16) Self Supervision

Autoencoders: $\vec{x} \to$ encoder $\to \vec{l} \to$ decoder $\to \hat{\vec{x}} \to$ surrogate obj

learn $\vec{l} \to$ neural network $\to \vec{y}$

↑ use larger set of unlabeled

Surrogate tasks:

vanilla autoencoding: naive approach, $L(\vec{x}_i, D(E(\vec{x}_i)))$

denoising autoencoding: $L(\vec{x}_i, D(E(\vec{x}_i + \vec{n})), \vec{n} \sim N(0, \sigma^2)$

masked autoencoding: $L(\vec{x}, D(E(\vec{x}')))\ \vec{x}' = [x_1, \dots x_3, x_4\ ?\ ]^T$

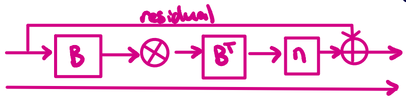
1) First Parameterization

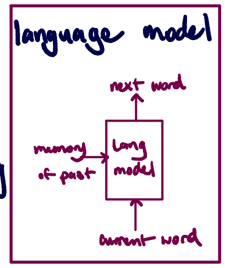
$\vec{x} \to \boxed{A} \to \vec{l} \to \boxed{B} \to \hat{\vec{x}}$

2) Second Parameterization

$\vec{x} \to \boxed{(B^T B)^{-1} B^T} \to \vec{l} \to \boxed{B} \to \hat{\vec{x}}$

3) Third Parameterization: RNN w/ weight sharing





Beam Search: keep bag of current best possibilities, choose k best, least negative, add them together

(17)(18)(19) Transformer Models

attention

1) drop recurrence from RNN, keep weight sharing
   1) soft appox hash table
   2) queryable softmax pooling

hash table pairs k_i and v_i, apply a query $\vec{q}$

$e_{i,t} = \frac{\langle \vec{q}_t, \vec{k}_i \rangle}{\sqrt{d}}$, $\alpha_{i,t} = \frac{e^{e_{i,t}}}{\sum_j e^{e_{j,t}}}$, output $= \sum \alpha_{i,t} \vec{v}_i$

2) can also use multiple channels in transformers, multi head attention query through multiple heads & concatenate outputs

- for a query, output is approx corresponds to nearest key, inner product $\mathrm{sim}(q,k) = \frac{\langle q, k \rangle}{\sqrt{d}}$
- attention mechanism to learn dependencies across input sequence, agg info, no learnable parameters

Transformers

1) inputs $x_{t,1}$
2) create key vector $k_{t,1}$, query vector $q_{t,1}$, value vector $v_{t,1}$
3) store key vector $k_{t,1}$, $v_{t,1}$ into table
4) Pass query vec $q_{t,1}$ as input to attention block, softmax
5) attention to linear layer W, combined with $x_{t,1}$ with skip connect
6) layer Norm $\to$ MLP $\to$ LN

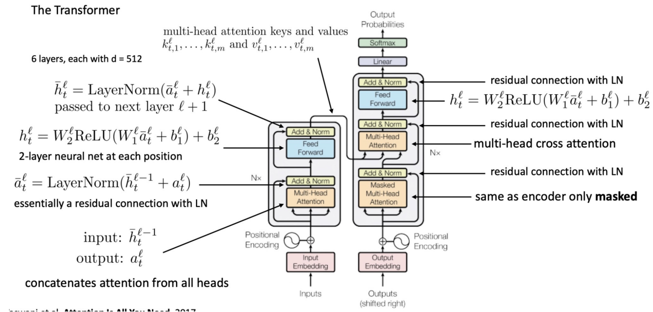
Masking: should not perform inner products w/ key-value pairs from future, set future values to 0

Cross-Attention: queries from decoder, key-value from encoder

Position Encoding: add positions to transformer input tokens concatenating position to vector or adding it

Tranformers: self attention layer, relevant to next, cross-attention w key-value from encoder, weight matrix has learnable weights

Represent words w/ embedding to show similarity



(20) Pretraining and fine-tuning

input $\to$ tokenizer (tokens $\to$ id) $\to$ lookup table (id $\to$ vector) $\to$ transformer

Word2vec: find vector representations of each word where similar words have similar embeddings

$\arg\max \sum_{c,o} \log \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$

↓

$\arg\max \sum (\log \sigma(u_o^T v_c)) + \sum \log \sigma(-u_w^T v_c))$

Pretrained models:

1. train language model on surrogate task
2. Run it on a sentence
3. Take hidden state from model and treat it as embedding

masked self-attention: make sure model doesn't just look ahead

BERT: bidirectional Transformer Language Models, mask percentage of inputs, don't need masked self-attention trained on predicting mask and next sentence prediction, to predict if two sentences swapped

Finetuning can either freeze BERT and retrain classifier at end or retrain end-to-end (genuine pretrained transformer)

GPT: autoregressive model, one directional transformer

(21)(22)(23) Fine tuning

Fine-tuning: 1) pretrain large model w/ self supervision & lots data
2) fine tune on task specific data or objective

Feature Extraction: decapitate training head and train new head

↑ less params to retrain, scalability, ↓ ok performance

Fine-tuning: replace head, retrain entire model

↑: increased capacity, ↓ overfitting, harder scale, divergent backbone grad

Prompt Engineering: prompts to make model do what we want

↑: no training, ↓ bad performance, limited training data

Prompt Tuning: create prompts in vector language

Catastrophic Forgetting: forgetting how to perform old tasks when trained to do new ones,

continual learning: learns series of tasks sequentially

- earlier layers are somewhat task specific
- Skip connections allow early layers to learn task specific

↳ Naive approach is to batch learn randomly

↳ **Replay during training:** examples of old task when training on new, (preferred solution)

↳ **Learning w/o Forgetting:** create pseudo labels after running through old heads, knowledge distillation, generate analog labels

T5/BART: BERT: encoder only, masked autoencoder
GPT: decoder only, predict next token

encoder-decoder transformers, masked autoencoder
mask spans tokens

Soft prompts: population of attention tables in between layers
- fine tuning can improve performance, but if model params too small, prompt cannot work

24 Meta Learning

Meta-learning: find system to quickly, reliably learn new tasks

1) Feature extract 2) fine tuning

1) randomly initialize task specific head
2) either fine tune entire model or just head
3) use sgd to update params w/ differentiable loss func

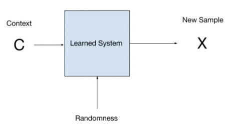
MAML ↓ exploding grad while training, memory

Alternatives: train on union of tasks

ANIL/Meta Opt Net/R2D2: freeze "feature extractor" optimize task head, differentiate w params

25 26 Generative Tasks

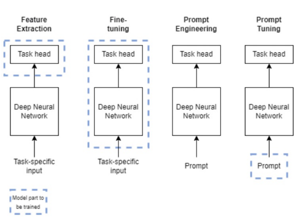
Generation Model: generate unseen example of data

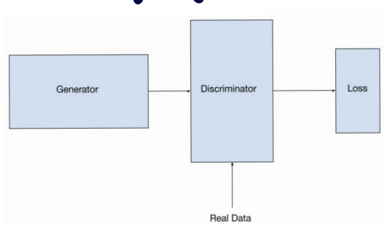


typically provide context (prefix) which is passed through layers

Generative Adversarial Networks: generator to create images and classifier (discriminator) to tell if image real or fake, loss trains both at same time.

↓ delicate, mode collapse if same output, always rejects



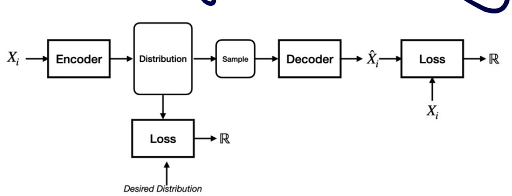


1) Train discriminator given frozen generator
2) Train generator given frozen discriminator, given real or fake either minimize or max sgd steps

Mode collapse: if generator lacks diversity

27 Diffusion

Variational Autoencoders: input to decoder random during training, add loss term on distribution of z, parameterize sgd, estimate density



Variational Autoencoders

VAEs fall in the class of likelihood-based models. The goal is to learn a model $p_\theta(x)$ that is close to the true distribution p_{data}. This is done by maximizing the likelihood of the observed data under the model, i.e. $\max_\theta \mathbb{E}_{x\sim p_{\text{data}}}\left[\log p_\theta(x)\right]$.

Evidence Lower Bound

Given a latent-variable model, $z \to x$, we have an alternative objective to maximize the likelihood of the data, i.e. $\max_\theta \mathbb{E}_{x\sim p_{\text{data}}}\left[\log p_\theta(x)\right]$. In particular, consider the expression

$$\log p_\theta(x) = \log \int_z p_\theta(x,z)\mathrm{d}z = \log \int_z p_\theta(x|z)p(z)\mathrm{d}z$$

- latent var z between encoder and decoder is noisy
- find probability $x_t|x_{t-1}$ as cond distribution, procedure for denoising

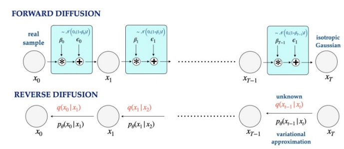


Figure 4: Diffusion Model iteratively adds noise to the input skewing the distribution towards well-understood distributions.

Diffusion models are a class of generative models that are *multi-step*, meaning that they perform multiple steps (say **T**) of inference. In particular, diffusion models are composed of two phases:

- **Forward**: In this phase, we start from samples from the real distribution and iteratively add noise to the samples. In particular, starting from $x_0 \sim p_{\text{data}}$ we have:

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\sqrt{\beta_{t-1}}\mathbf{x}_{t-1}, (1-\beta_{t-1})\mathbb{I})$$

Notably, the variance of the distribution is a function of the step t, and the variance increases as we move forward in time. A typical example [3] of this would be linearly increasing from $\beta_0 = 10^{-4}$ to $\beta_T = 0.02$.

- **Reverse**: Sampling from the diffusion model is done by starting from a sample from our target distribution (e.g. isotropic Gaussian) and iteratively denoising the noisy inputs. In the Markov chain illustrated in fig. 4 this corresponds to starting at x_T and moving backwards in time, such that x_0 is the denoised sample.

Performing this denoising step, however requires us to estimate $q(x_{t-1}|x_t)$ which is not tractable. Instead, we perform a variational

[3] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *Advances in Neural Information Processing Systems*, 33: 6840–6851, 2020

Disc

- robotic navigation task
  - encoder/decoder (T5), encode target, inputs history of trajectory in decoder, auto regressive masking, output three logits, crossentropy loss
- subword tokenizers preferred, word tokenizers map words not in vocab to <UNK>

175B param, 100k data → soft prompting
90B param, no data classification → hard prompting
1B param, 100 data image classify → feature extraction
1B param, 100M data ImageNet → full finetuning