Assorted Topics (1)

CS772A: Probabilistic Machine Learning

Piyush Rai

Plan Today

- Some other deep generative models
  - Large Language Models
  - Diffusion Models for text
- Model Calibration
  - How to prevent models from being "overconfident"
- Frequentist statistics
  - Another way to get a distribution over parameters without being Bayesian

Large Language Models (LLM)

- An LLM defines a probability distribution over sequences of tokens

$$\boldsymbol{x} = \{x_1, x_2, \ldots, x_N\}$$

- Autoregressive modeling is a popular way to define this distribution

$$p(\boldsymbol{x}) = p(x_1)p(x_2|x_1)p(x_3|x_1, x_2) \ldots = \prod_{i=1}^{N} p(x_i|\boldsymbol{x}_{<i})$$

- Params $\boldsymbol{\theta}$ of distribution $p(x_i|\boldsymbol{x}_{<i})$ defined using a neural net (e.g., transformer)

$$p_\theta(x_i|\boldsymbol{x}_{<i}) = \text{softmax}(f_\theta(\boldsymbol{x}_{<i}))$$

Vector of probabilities of all possible values of the next token

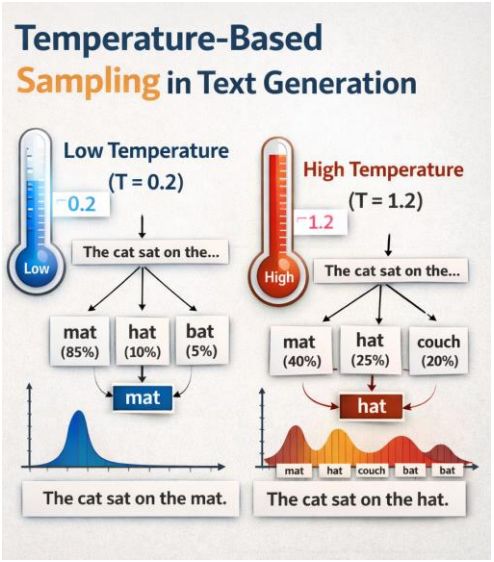
A neural net

Training of LLMs and Sequence Generation

- Usually trained using maximum likelihood with log-likelihood defined as

$$\mathcal{L}(\theta) = \sum_{i=1}^{N} \log p_\theta(x_i | \boldsymbol{x}_{<i})$$

- Once trained, generate a sequence of tokens, one at a time. Some popular ways:
  - Greedy (pick the most probable token deterministically): $\hat{x}_i = \text{argmax } p_\theta(x_i | \boldsymbol{x}_{<i})$
  - Sampling: $\hat{x}_i \sim p_\theta(x_i | \boldsymbol{x}_{<i})$
  - Temperature based sampling: $\hat{x}_i \sim [p_\theta(x_i | \boldsymbol{x}_{<i})]^{1/T}$
    - $T < 1$ sharpens the distribution (more deterministic sampling)
    - $T > 1$ flattens the distribution (more exploratory sampling)
  - Top-k sampling
    - Randomly sample a token from k most probable token
  - Nucleus (top-p) sampling
    - Sample from minimum set of tokens with cumulative probability $\geq p$



Some Limitations of Autoregressive LLMs

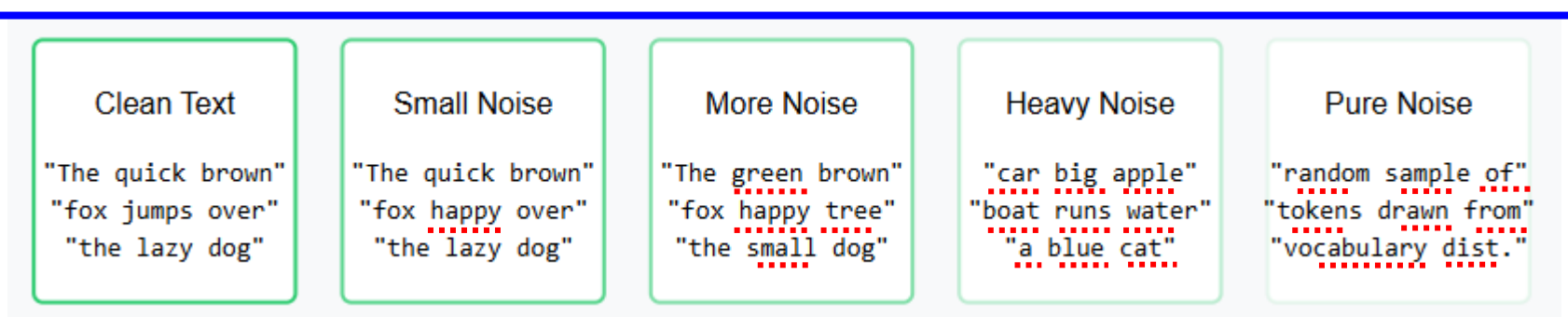
- Sequential Generation: Inherently slow due to token-by-token decoding
- Low Output Diversity: Because of the decoding techniques used
- Locally greedy generation and lacks long-term coherence control.
- Token-Level Objectives: Next-token prediction doesn't align well with task-level goals (e.g., factual consistency).
- Difficulty Handling Edits/Rewrites: Inefficient for tasks requiring partial edits or structured generation.

Diffusion based LLM*

- Autoregressive LLMs generate each token conditioned on earlier tokens

$$p(\boldsymbol{x}) = \prod_{i=1}^{N} p(x_i | x_{<i})$$

- In contrast, diffusion based LLM generate all tokens in parallel
- Diffusion LLM consist of a forward and a reverse process
- Forward process corrupts the token sequence gradually till it becomes pure noise



- Reverse process starts with pure noise and gradually denoises it to generates a token sequence

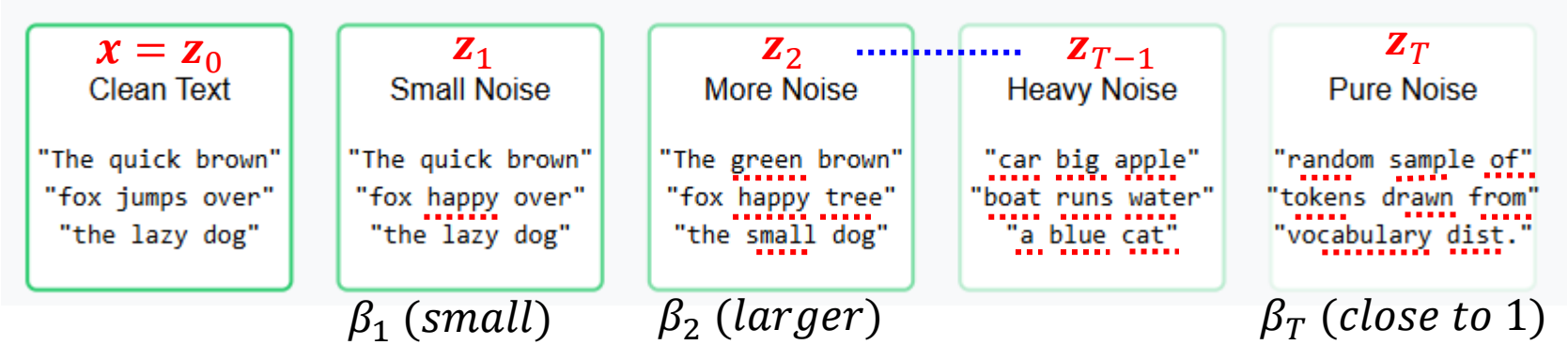
*Structured Denoising Diffusion Models in Discrete State-Spaces (Austin et al, NeurIPS 2021)
*Argmax Flows and Multinomial Diffusion: Learning Categorical Distributions (Hoogeboom et al, NeurIPS 2021)

Forward Process

- Assuming z_t contains N tokens, the forward process in diffusion LLM can be defined as

$$q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}) = \prod_{i=1}^{N} \text{Cat}(z_t^i|\boldsymbol{P}z_{t-1}^i)$$

$\boldsymbol{P}$ is the $V \times V$ transition matrix that defines corruption probabilities, z_t^i are one-hot vectors of size V where V is the vocab size



- A very simple yet popular form of the above corruption distribution is

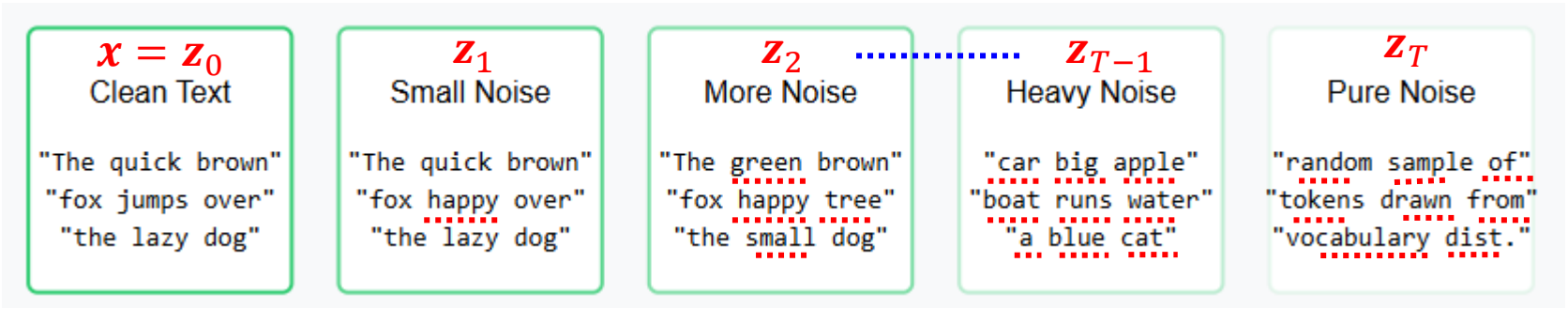
$$q(\boldsymbol{z}_t|\boldsymbol{z}_{t-1}) = (1-\beta_t)\mathbb{I}[\boldsymbol{z}_t = \boldsymbol{z}_{t-1}] + \beta_t/V$$

- Basically, to get sequence $\boldsymbol{z_t}$ from $\boldsymbol{z_{t-1}}$, it does the following for each token in $\boldsymbol{z_{t-1}}$
  - With probability $\boldsymbol{\beta_t}$, replace it by a random token from the vocabulary
  - With probability $\boldsymbol{1-\beta_t}$, keep it unchanged
- Note: Some diffusion LLMs replace tokens by not a random but a "mask" token

Reverse Process

- Takes noisy text and produces less noisy text (basically opposite of forward process)

$$p_\theta(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t) = \prod_{i=1}^{N} \text{Cat}(z_{t-1}^i | p_\theta(z_{t-1}^i | z_t^i))$$



- The training objective is similar to the one used in continuous data LLM
  - Basically we want to match $p_\theta(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t)$ and $q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x})$

$$\mathcal{L} = \mathbb{E}_{t,\boldsymbol{x},\boldsymbol{z}_t}[\|p_\theta(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t) - q(\boldsymbol{z}_{t-1}|\boldsymbol{z}_t, \boldsymbol{x})\|^2]$$

Diffusion LLMs: Some Pros and Cons

- Some pros
  - Parallel Decoding ⟶ Faster inference potential via non-sequential generation
  - Better Output Diversity ⟶ Naturally handles multi-modal distributions
  - Improved Controllability ⟶ Supports classifier-free guidance and conditioning
  - Resilience to Exposure Bias ⟶ Trained via denoising, not next-token prediction
  - Flexible Objectives ⟶ Enables structured generation, editing, and planning
- Some cons
  - Slower Training ⟶ Iterative denoising steps can increase training cost
  - Complex Architecture ⟶ Needs noise schedule, denoising network, sampling strategy
  - High Inference Cost (currently) ⟶ Requires multiple denoising steps at test time
  - Less Mature ⟶ Fewer benchmarks and toolkits compared to autoregressive LLMs
  - Tokenization Challenges ⟶ Needs careful handling of discrete text representations

Model Calibration

Calibration

- Assume a classifier that outputs probabilities $f(x_n) = [a_{n1}, a_{n2}, \ldots, a_{nC}]$ such that

Predicted label

$$\hat{y}_n = \text{argmax}_{c=\{1,2,\ldots,C\}}\, a_{nc}$$

Probability of the predicted label (confidence of f for this prediction)

$$\hat{a}_n = \max_{c=\{1,2,\ldots,C\}} a_{nc}$$

- Notion of calibration: Predictions should not neither be over-confident, nor under-confident
- Desirable: Predictions with confidence $\mu \in (0,1)$ are correct $(100 \times \mu)\%$ of the time
- Assume $\mathcal{B}_b$ as set of samples for which $\hat{a}_n$ falls in bin $I_b = (\frac{b-1}{B}, \frac{b}{B}]$

Average accuracy of bin b

$$\text{acc}(B_b) = \frac{1}{|B_b|} \sum_{n \in B_b} \mathbb{I}(\hat{y}_n = y_n)$$

Average confidence of bin b

$$\text{conf}(B_b) = \frac{1}{|B_b|} \sum_{n \in B_b} \hat{a}_n$$

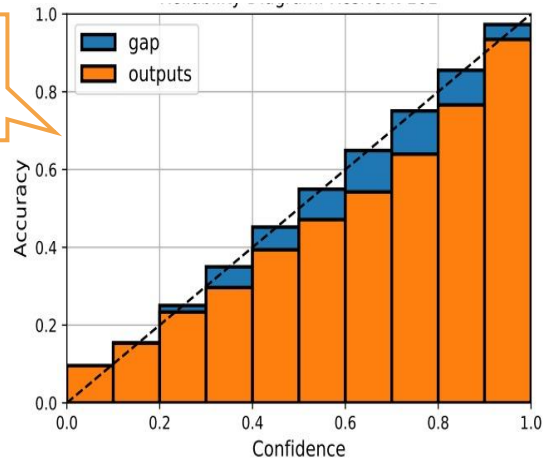
- We want bins' average accuracies to match bins' average confidence

Reliability Diagrams and A Calibration Metric

- Reliability diagrams are plots of accuracy vs confidence

Using $B = 10$ equal-width bins

It's just one simple way; other ways also possible to construct the bins



- Several metrics exist to measure how well-calibrated the model's predictions are
- Expected Calibration Error (ECE) is one such popular metric

Should be small for a well-calibrated model

$$\mathrm{ECE}(f) = \sum_{b=1}^{B} \frac{|\mathcal{B}_b|}{B} \left|\mathrm{acc}(\mathcal{B}_b) - \mathrm{conf}(\mathcal{B}_b)\right|$$

ECE is the average "gap" area in the reliability diagram

Calibration Methods (contd)

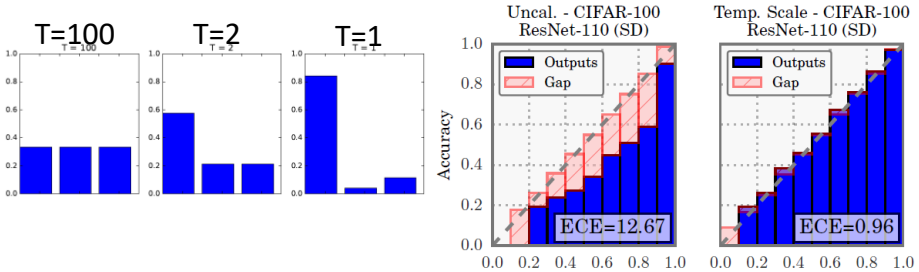
Parameters of the trained model are kept frozen in this process

The scaling parameters (w or T) are learned by minimizing the loss on some validation set.

- Method 1: Calibrate an already trained model in a post-hoc manner, e.g.,
  - Requires learning to scale the logits produced by the model, e.g.,

$$\text{softmax}(z_1, z_2, \ldots, z_C) \Rightarrow \text{softmax}\,(w_1 z_1 + b_1, w_2 z_2 + b_2, \ldots, w_C z_C + b_c)$$

$$\text{softmax}(z_1, z_2, \ldots, z_C) \Rightarrow \text{softmax}\left(\frac{z_1}{T}, \frac{z_2}{T}, \ldots, \frac{z_C}{T}\right)$$



- Method 2: Change the training procedure, e.g.,
  - Add a regularizer which avoids overconfident predictions

Maximize the likelihood

Maximize the entropy of the predictive distribution to reduce overconfidence

$$\mathcal{L} = \sum_{i=1}^{N} \log p(y_i|x_i, w) + \mathbb{H}[\log p(y_i|x_i, w)]$$

  - Trained with smoothed labels instead of one-hot labels

$$[0, 0, 1, 0] \Rightarrow [0.05, 0.05, 0.85, 0.05]$$

Frequentist Statistics
(vs Bayesian Statistics)

Frequentist Statistics

- The Bayesian approach treats parameters/model unknowns as random variables
- In the Bayesian approach, the posterior over these r.v.'s help capture the uncertainty
- The Frequentist approach is a different way to capture uncertainty
  - Don't treat parameters as r.v. but as fixed unknowns
  - Treat parameters as a function of the dataset, e.g., $\hat{\theta}(\mathcal{D}) = \pi(\mathcal{D})$

This can be some point estimate, e.g., MLE, MAP, method of moments, etc.

  - Variations in param estimates over different datasets represents their uncertainty

A random dataset drawn from the true data distribution

True unknown value of the parameter

$$\tilde{\mathcal{D}}^{(s)} = \{\boldsymbol{x}_n \sim p(\boldsymbol{x}_n|\boldsymbol{\theta}^*) : n = 1 : N\} \qquad (s = 1, 2, \ldots, S)$$

The estimated distribution of the parameters given any randomly drawn dataset from the true data distribution

$$p(\pi(\tilde{\mathcal{D}}) = \boldsymbol{\theta}|\tilde{\mathcal{D}} \sim \boldsymbol{\theta}^*) \approx \frac{1}{S}\sum_{s=1}^{S} \delta(\boldsymbol{\theta} = \pi(\tilde{\mathcal{D}}^{(s)}))$$

Param estimate using the s-th sampled dataset

As $S \to \infty$, this is known as the "sampling distribution" of the estimator

Note that sampling distribution is different from a posterior distribution we infer in Bayesian learning (there, we condition on a fixed training set)

But if the estimator is MLE and Bayesian method's prior is uniform, then both distributions are very similar (sampling distribution is often called "poor man's posterior"

Approximating the sampling distribution

- Since the true $\boldsymbol{\theta}^*$ is not known, we can't compute the sampling distribution exactly

$$\tilde{\mathcal{D}}^{(s)} = \{\boldsymbol{x}_n \sim p(\boldsymbol{x}_n|\boldsymbol{\theta}^*) : n = 1 : N\} \qquad (s = 1, 2, \ldots, S)$$

$$p(\pi(\tilde{\mathcal{D}}) = \boldsymbol{\theta}|\tilde{\mathcal{D}} \sim \boldsymbol{\theta}^*) \approx \frac{1}{S}\sum_{s=1}^{S} \delta(\boldsymbol{\theta} = \pi(\tilde{\mathcal{D}}^{(s)}))$$

- Bootstrap is a popular method to approximate the sampling distribution
- Two types of bootstrap methods: parametric and nonparametric bootstrap

Parametric Bootstrap

- Get a point est. of $\boldsymbol{\theta}$ using training data

$$\hat{\boldsymbol{\theta}} = \pi(\mathcal{D})$$

- Generate multiple datasets using $\hat{\boldsymbol{\theta}}$ as

$$\tilde{\mathcal{D}}^{(s)} = \{\boldsymbol{x}_n \sim p(\boldsymbol{x}_n|\hat{\boldsymbol{\theta}}) : n = 1 : N\} \quad (s = 1, 2, \ldots, S)$$

- Now compute the approximation as

$$p(\pi(\tilde{\mathcal{D}}) = \boldsymbol{\theta}|\tilde{\mathcal{D}} \sim \boldsymbol{\theta}^*) \approx \frac{1}{S}\sum_{s=1}^{S} \delta(\boldsymbol{\theta} = \pi(\tilde{\mathcal{D}}^{(s)}))$$

Nonparametric Bootstrap

- Use sampling with replacement on original training set to generate S datasets with N datapoints in each

Each dataset will contain roughly 63% unique datapoints from original training set

- Now compute the approximation as

$$p(\pi(\tilde{\mathcal{D}}) = \boldsymbol{\theta}|\tilde{\mathcal{D}} \sim \boldsymbol{\theta}^*) \approx \frac{1}{S}\sum_{s=1}^{S} \delta(\boldsymbol{\theta} = \pi(\tilde{\mathcal{D}}^{(s)}))$$